Neural Network Perception for Mobile Robot Guidance

Dean A. Pomerleau

February 16, 1992
CMU-CS-92-115

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Support for this work has come from DARPA, under contracts DACA76-85-C-0019, DACA76-85-C-0003, DACA76-85-C-0002, DACA76-89-C-0014 and DAAE07-90-C-R059. These contracts were monitored by the Topographic Engineering Center and by TACOM. This research was also funded in part by grants from the Fujitsu Corporation and the Shimizu Corporation.

Neural Network Perception for Mobile Robot Guidance

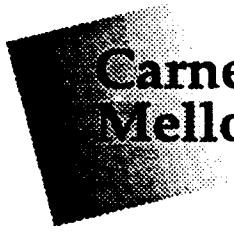
Dean A. Pomerleau

February 16, 1992
CMU-CS-92-115

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Support for this work has come from DARPA, under contracts DACA76-85-C-0019, DACA76-85-C-0003, DACA76-85-C-0002, DACA76-89-C-0014 and DAAE07-90-C-R059. These contracts were monitored by the Topographic Engineering Center and by TACOM. This research was also funded in part by grants from the Fujitsu Corporation and the Shimizu Corporation.



School of Computer Science

DOCTORAL THESIS
in the field of
Computer Science

Neural Network Perception for Mobile Robot Guidance

DEAN POMERLEAU

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

ACCEPTED:

David S. Touretzky — 2/10/92

MAJOR PROFESSOR — **DATE**

R. Rutenbar — 2/17/92

DEAN — **DATE**

APPROVED:

[signature] — 2/18/92

PROVOST — **DATE**

Dedicated to Terry, Glen and Phyllis

Abstract

Vision based mobile robot guidance has proven difficult for classical machine vision methods because of the diversity and real time constraints inherent in the task. This thesis describes a connectionist system called ALVINN (Autonomous Land Vehicle In a Neural Network) that overcomes these difficulties. ALVINN learns to guide mobile robots using the back-propagation training algorithm. Because of its ability to learn from example, ALVINN can adapt to new situations and therefore cope with the diversity of the autonomous navigation task.

But real world problems like vision based mobile robot guidance presents a different set of challenges for the connectionist paradigm. Among them are:

- How to develop a general representation from a limited amount of real training data,
- How to understand the internal representations developed by artificial neural networks,
- How to estimate the reliability of individual networks,
- How to combine multiple networks trained for different situations into a single system,
- How to combine connectionist perception with symbolic reasoning.

This thesis presents novel solutions to each of these problems. Using these techniques, the ALVINN system can learn to control an autonomous van in under 5 minutes by watching a person drive. Once trained, individual ALVINN networks can drive in a variety of circumstances, including single-lane paved and unpaved roads, and multi-lane lined and unlined roads, at speeds of up to 55 miles per hour. The techniques also are shown to generalize to the task of controlling the precise foot placement of a walking robot.

Acknowledgements

I wish to thank my advisor, Dr. David Touretzky for his support and technical advice during my graduate studies. Dave has not only provided invaluable feedback, he has also given me the latitude I've needed to develop as a researcher. I am also grateful to Dr. Charles Thorpe for the opportunities, resources and expertise he has provided me. Without Chuck's support, this research would not have been possible. I also wish to thank the other members of my committee, Dr. Takeo Kanade and Dr. Terrence Sejnowski, for their insightful analysis and valuable comments concerning my work.

I owe much to all the members of the ALV/UGV project. Their technical support and companionship throughout the development of the ALVINN system has made my work both possible and enjoyable. I would like to specifically thank Jay Gowdy and Omead Amidi, whose support software underlies much of the ALVINN system. James Frazier also deserves thanks for his patience during many hours of test runs on the Navlab.

Interaction with members of the Boltzmann group at Carnegie Mellon has also been indispensable. From them I have not only learned about all aspects of connectionism, but also how to communicate my thoughts and ideas. In particular, the insights and feedback provided by discussions with John Hampshire form the basis for much of this thesis. I am grateful to Dave Plaut, whose helpful suggestions in the early stages of this work put me on the right track.

Other people who have contributed to the success of this thesis are the members of the SM^2 group. In particular, Ben Brown and Hiroshi Ueno have given me the opportunity, incentive and support I have needed to explore an alternative domain for connectionist mobile robot guidance.

I am also in debt to my office mates, Spiro Michaylov and Nevin Heintze. They have helped me throughout our time as graduate students, with everything from explaining LaTeXpeculiarities to feeding my fish. I would like to thank my parents, Glen and Phyllis, for encouraging my pursuit of higher education, and for all the sacrifices they've made to provide me with a world of opportunities. Finally, I am especially grateful to my fiancée, Terry Jessie, for her constant love, support and patience during the difficult months spent preparing this dissertation. Her presence in my life has helped me keep everything in perspective.

Dean A. Pomerleau

February 16, 1992

Contents

Chapter 1

Introduction

A truly autonomous robot must sense its environment and react appropriately. Previous mobile robot perception systems have relied on hand-coded algorithms for processing sensor information. In this dissertation I develop techniques which enable artificial neural networks (ANNs) to learn the visual processing required for mobile robot guidance. The power and flexibility of these techniques are demonstrated in two domains, wheeled vehicle navigation, and legged robot foot positioning.

The central claims of this dissertation are:

- By appropriately constraining the problem, the network architecture and the training algorithm, ANNs can quickly learn to perform many of the complex perception tasks required for mobile robot navigation.

- A neural network-based mobile robot perception system is able to robustly handle a wider variety of situations than hand-programmed systems because of the ability of ANNs to adapt to new sensors and situations.

- Artificial neural networks are not just black boxes. Their internal representations can be analyzed and understood.

- The reliability of ANNs can be estimated with a relatively high precision. These reliability estimates can be employed to arbitrate between multiple expert networks, and hence facilitate the modular construction of connectionist systems.

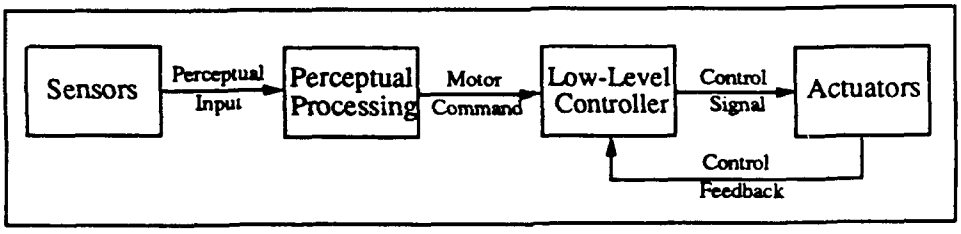


Figure 1.1: Block diagram of sensor based mobile robot guidance

- By combining neural network-based perception with symbolic reasoning, an autonomous navigation system can achieve accurate low-level control *and* exhibit intelligent high-level behavior.

1.1 Problem Description

To function effectively, an autonomous mobile robot must first be capable of purposeful movement in its environment. This dissertation focuses on the problem of how to employ neural network based perception to guide the movement of such a robot. To navigate effectively in a complex environment, a mobile robot must be equipped with sensors for gathering information about its surroundings. In this work, I have chosen imaging sensors as the primary source of information because of their ability to quickly provide dense representations of the environment. The imaging sensors actually employed include color and black-and-white video cameras, a scanning laser rangefinder and a scanning laser reflectance sensor[1].

The imaging sensors provide input to the component of an autonomous mobile robot which will be the primary focus of this dissertation, the perceptual processing module (See Figure 1.1). The job of the perceptual processing module is to transform the information about the environment provided by one or more imaging sensors into an appropriate high level motor command. The motor command appropriate for a given situation depends both on the current state of the world as reported by the sensors, and the perception module's knowledge of appropriate responses for particular situations. The motor responses produced by the perception module take the form of elementary movement directives, such as "drive the robot

[1] Work is also underway in using the same techniques to interpret the output from a sonar array and an infrared camera.

along an arc with a 30m radius" or "move the robot's foot 2.5cm to the right".

The elementary movement directives are carried out by a controller which manipulates the robot's actuators. The determination of the correct motor torques required to smoothly and accurately perform the elementary movement directives is not addressed in this dissertation. While it is possible to use connectionist techniques for low level control [Jordan & Jacobs, 1990, Katayama & Kawato, 1991], this aspect of the problem is implemented using classical PID control in each of the systems described in this work.

The two mobile robot domains used to develop and demonstrate the techniques of this thesis are autonomous outdoor driving and precise foot positioning for a robot designed to walk on the exterior of a space station. Because it embodies most of the difficulties inherent in any mobile robot task, autonomous outdoor driving is the primary focus of this work.

In autonomous outdoor driving, the goal is to safely guide the robot through the environment. Most commonly, the environment will consist of a network of roads with varying characteristics and markings. In this situation, the goal is to keep the robot on the road, and in the correct lane when appropriate. There is frequently the added constraint that a particular route should be followed through the environment, requiring the system to make decisions such as which way to turn at intersections. An additional desired behavior for the autonomous robot is to avoid obstacles, such as other cars, when they appear in its path.

The difficulty of outdoor autonomous driving stems from four factors. They are

- Task variations due to changing road type
- Appearance variations due to lighting and weather conditions
- Real time processing constraints
- High level reasoning requirements

A general autonomous driving system must be capable of navigating in a wide variety of situations. Consider some of the many driving scenarios people encounter every day: There are multi-lane roads with a variety of lane markers. There are two-lane roads without lane markers. There are situations, such as city or parking lot driving, where the primary guidance comes not from the road delineations, but from the need to avoid other cars and pedestrians.

The second factor making autonomous driving difficult is the variation in appearance that results from environmental factors. Lighting changes, and deep shadows make it difficult for perception systems to consistently pick out important features during daytime driving. The low light conditions encountered at night make it almost impossible for a video-based system to drive reliably. In addition, missing or obscured lane markers make driving difficult for an autonomous system even under favorable lighting conditions.

Given enough time, a sophisticated image processing system might be able to overcome these difficulties. However the third challenging aspect of autonomous driving is that there is a limited amount of time available for processing sensor information. To blend in with traffic, an autonomous system must drive at a relatively high speed. To drive quickly, the system must react quickly. For example, at 50 miles per hour a vehicle is traveling nearly 75 feet per second. A lot can happen in 75 feet, including straying a significant distance from the road, if the system isn't reacting quickly or accurately enough.

Finally, an autonomous driving system not only must perform sophisticated perceptual processing, it also must make high level symbolic decisions such as which way to turn at intersections. This dissertation shows that the first three factors making mobile robot guidance difficult can be overcome using artificial neural networks for perception, and the fourth can be handled by combining artificial neural networks with symbolic processing.

1.2 Robot Testbed Description

The primary testbed for demonstrating the applicability of the ideas developed in this dissertation to autonomous outdoor driving is the CMU Navlab, shown in Figure 1.2. The Navlab is a modified Chevy van equipped with forward and backward facing color cameras and a scanning laser rangefinder for sensing the environment. These are the primary sensory inputs the system receives. The Navlab also contains an inertial navigation system (INS) which can maintain the vehicle's position relative to its starting location. The Navlab is outfitted with three Sun Sparcstations, which are used for perceptual processing and other high level computation. The Navlab also has a 68020-based processor for controlling the steering wheel and accelerator and for monitoring the vehicle's status.

The Navlab can be controlled by computer or driven by a person just like a normal car. This human controllability is useful for getting the Navlab to a test site,

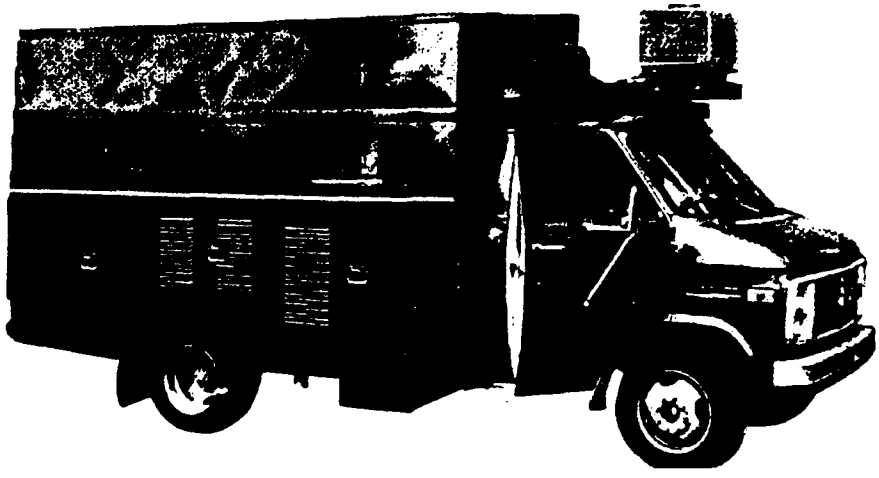


Figure 1.2: The CMU Navlab Autonomous Navigation Testbed

and as will be seen in Chapter 3, for teaching an artificial neural network to drive by example. I have used two other robots to demonstrate the power of connectionist robot guidance, a ruggedized version of the Navlab called Navlab II, and a walking robot called the Self Mobile Space Manipulator (SM^2). These additional testbeds will be described in more detail in Chapters 5 and 9, respectively.

1.3 Dissertation Overview

The goal of this thesis is to develop techniques that enable artificial neural networks to guide mobile robots using visual input. In chapter 2, I present the simple neural network architecture that serves as the basis for the connectionist mobile robot guidance system I develop called ALVINN (Autonomous Land Vehicle In a Neural Network). The architecture consists of a single hidden layer, feedforward network (see Figure 1.3). The input layer is a two dimensional retina which receives input from an imaging sensor such as a video camera or scanning laser rangefinder. The output layer is a vector of units representing different steering responses, ranging from a sharp left to a sharp right turn. The network receives as input an image of the road ahead, and produces as output the steering command that will keep the vehicle on the road.

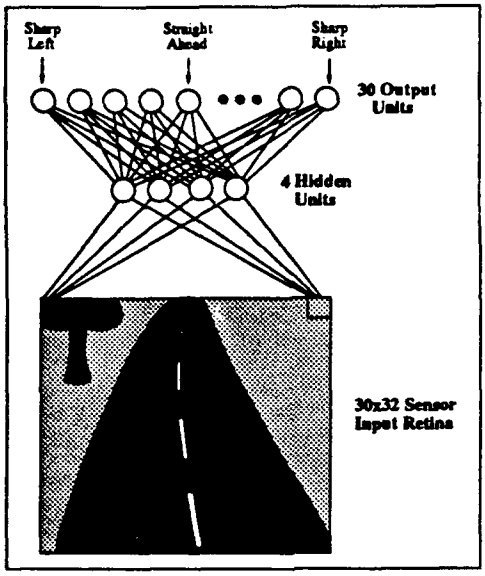


Figure 1.3: ALVINN driving network architecture

Although some aspects of ALVINN's architecture and I/O representation are unique, the network structure is not the primary reason for ALVINN's success. Instead, much of its success can be attributed to the training methods presented in Chapters 3 and 4. Using the training "on-the-fly" techniques described in Chapter 3, individual three-layered ALVINN networks can quickly learn to drive by watching a person steer. These methods allow ALVINN to learn about new situations first hand, as a person drives the vehicle. The ability to augment the limited amount of live training data available from the sensors with artificial images depicting rare situations is shown to be crucial for reliable network performance in both Chapters 3 and 4.

Using the architecture described in Chapter 2, and the training techniques from Chapters 3 and 4, ALVINN is able to drive in a wide variety of situations, described in Chapter 5. As a preview, some of ALVINN's capabilities include driving on single-lane paved and unpaved roads, and multi-lane lined and unlined roads, at speeds of up to 55 miles per hour.

But developing networks that can drive is not enough. It is also important to understand *how* the networks perform their functions. In order to quantitatively understand the internal representation developed by individual driving network, I

develop a technique called sensitivity analysis in Chapter 6. Sensitivity analysis is a graphical technique which provides insight into the processing performed by a network's individual hidden units, and into the cooperation between multiple hidden units to carry out a task. The analysis techniques in Chapter 6 illustrate that ALVINN's internal representation varies widely depending on the situations for which it is trained. In short, ALVINN develops filters for detecting image features that correlate with the correct steering direction.

A typical filter developed by ALVINN is shown in Figure 1.4. It depicts the connections projecting to and from a single hidden unit in a network trained on video images of a single-lane, fixed width road. This hidden unit receives excitatory connections (shown as white spots) from a road shaped region on the left of the input retina (see the schematic). It makes excitatory connections to the output units representing a sharp left turn. This hidden unit is stimulated when a road appears on the left, and suggests a left turn in order to steer the vehicle back to the road center. Road-shaped region detectors such as this are the most common type of feature filters developed by networks trained on single-lane unlined roads. In contrast, when trained for highway driving ALVINN develops feature detectors that determine the position of the lane markers painted on the road.

This situation specificity allows individual networks to learn quickly and drive reliably in limited domains. However it also severely limits the generality of individual driving networks. Chapters 7, 8 and 9 focus on techniques for combining multiple simple driving networks into a single system capable of driving in a wide variety of situations. Chapter 7 describes rule-based techniques for integrating multiple networks and a symbolic mapping system. The idea is to use a map of the environment to determine which situation-specific network is appropriate for the current circumstances. The symbolic mapping module is also able to provide ALVINN with something the networks lack, namely the ability to make high level decision such as which way to turn at intersections.

However rule-based arbitration is shown to have significant shortcomings. Foremost among them is that it requires detailed symbolic knowledge of the environment, which is often difficult to obtain. In Chapters 8 and 9, I develop connectionist multi-network arbitration techniques to complement the rule-based methods of Chapter 7. These techniques allow individual networks to estimate their own reliability in the current situation. These reliability estimates can be used to weight the responses from multiple networks and to determine when a new network needs to be trained.

Chapter 10 illustrates the flexibility of connectionist mobile robot guidance

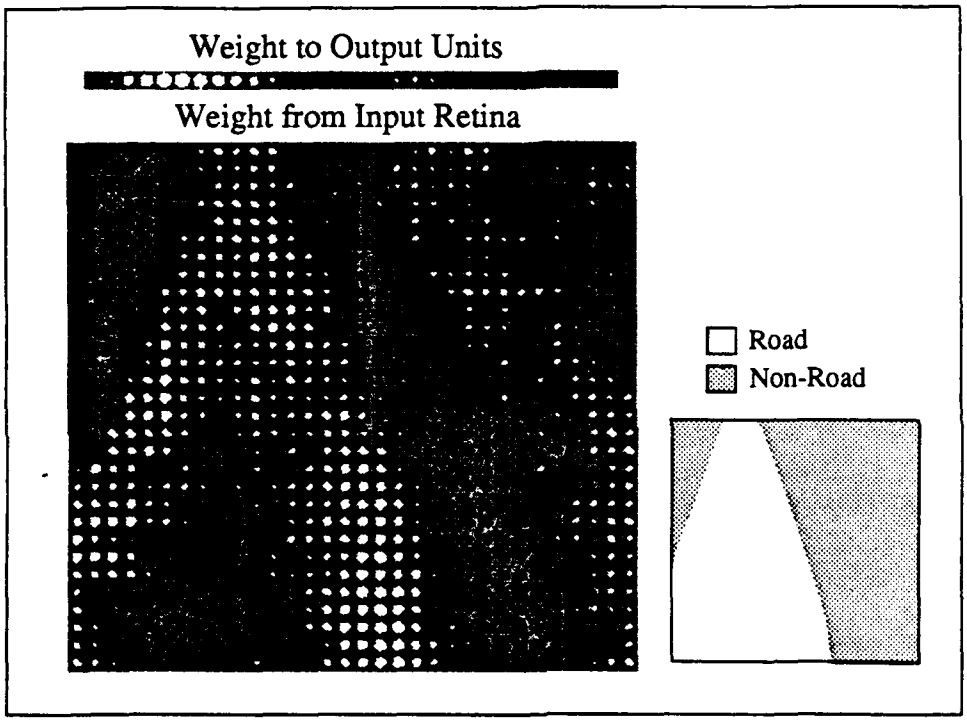


Figure 1.4: Diagram of weights projecting to and from a typical hidden unit in a network trained on roads with a fixed width. This hidden unit acts as a filter for a road on the left side of the visual field as illustrated in the schematic.

by demonstrating its use in a very different domain, the control of a two-legged walking robot designed to inspect the space station exterior. The crucial task in this domain is to precisely position the foot of the robot in order to anchor it without damaging either the space station or the robot. The same methods developed to steer an autonomous vehicle are employed to safely guide the foot placement of this walking robot.

In Chapter 11, the neural network approach to autonomous robot guidance is compared with other techniques, including hand-programmed algorithms and other machine learning methods. Because of its ability to adapt to new situations, ALVINN is shown to be more flexible than previous hand-programmed systems for mobile robot guidance. The connectionist approach employed in ALVINN is also demonstrated to have distinct advantages over other machine learning techniques such as nearest neighbor matching, decision trees and genetic algorithms.

Finally, Chapter 12 summarizes the results and discusses the contributions of this dissertation. It concludes by presenting areas for future work.

Chapter 2

Network Architecture

The first steps in applying artificial neural networks to a problem involve choosing a training algorithm and a network architecture. The two decisions are intimately related, since certain training algorithms require, or are best suited to, specific network architectures. For this work, I chosen a multi-layered perceptron (MLP) and the back-propagation training algorithm [Rumelhart, Hinton & Williams, 1986] for the following reasons:

- The task requires supervised learning from examples (i.e. given sensor input, the network should respond with a specific motor response). This rules out unsupervised/competitive learning algorithms like Kohonen's self-organizing feature maps [Kohonen, 1990] which learn to classify inputs on the basis of statistically significant features, but not to produce particular desired responses.

- The system should learn relatively quickly, since one of the goals is to rapidly adapt to new driving situations. This rules out certain supervised training algorithms/architectures such as Boltzmann Machines [Hopfield, 1982], which are notoriously slow at learning.

- The task of determining the correct motor response from sensor input was not expected to require substantial, run-time knowledge about recent inputs. Thus, it was decided that the extensions of the back-propagation algorithm to fully recurrent networks [Pineda, 1987, Pearlmutter, 1988] was not necessary.

The decision to use artificial neural networks in the first place, and to use back-propagation over other closely related neural network training algorithms like quickprop [Fahlman, 1988] and radial basis functions [Poggio & Girosi, 1990] can be better understood after presentation of the architecture and training scheme actually employed in this work, and hence will be discussed in Chapter 11.

Once the decision is made to use a feedforward multi-layered perceptron as the underlying network architecture, the question then becomes "what form should the MLP take?" This question can be divided into three components: the input representation, the output representation, and the network's internal structure. I will discuss each of the three components separately, theoretically and/or empirically justifying the choices made.

2.1 Architecture Overview

The architecture of the perception networks chosen for mobile robot guidance consists of a multi-layer perceptron with a single hidden layer (See Figure 2.1). The input layer of the network consists of a 30x32 unit "retina" which receives images from a sensor. Each of the 960 units in the input retina is fully connected to the hidden layer of 4 units, which in turn is fully connected to the output layer. The output layer represents the motor response the network deems appropriate for the current situation. In the case of a network trained for autonomous driving, the output layer consists of 30 units and is a linear representation of the direction the vehicle should steer in the current situation. The middle output unit represents the "travel straight ahead" condition, while units to the left and right of center represent successively sharper left and right turns.

To control a mobile robot using this architecture, input from one of the robot's sensors is reduced to a low-resolution 30x32 pixel image and projected onto the input retina. After completing a forward pass through the network, a motor response is derived from the output activation levels and performed by the low level controller. In the next sections, I will expand this high level description, giving more details of how the processing actually proceeds and why this architecture was chosen.

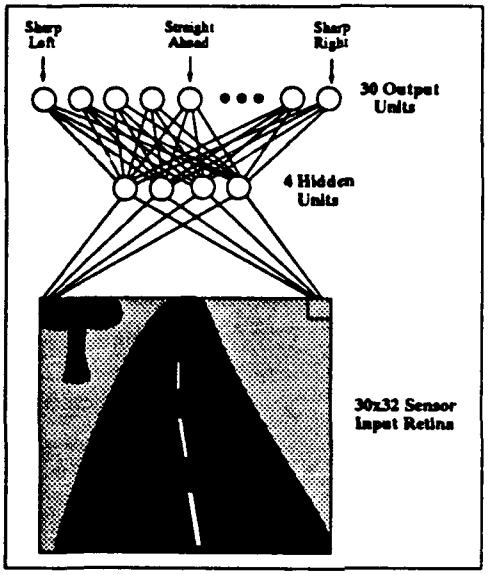


Figure 2.1: ALVINN driving network architecture

2.2 Input Representations

Perhaps the most important factor determining the performance of a particular neural network architecture on a task is the input representation. In determining the input representation, there are two schools of thought. The first believes that the best input representation is one which has been extensively preprocessed to make "important" features prominent and therefore easy for the network to incorporate in its processing. The second school of thought contends that it is best to give the network the "raw" input and let it learn from experience what features are important.

Four factors to consider when deciding the extent of preprocessing to perform on the input are the existence of a preprocessing algorithm, its necessity, its complexity and its generality. If straightforward algorithms are known to perform crucial initial processing steps applicable in a wide variety of situations, then it is advisable to use them. Such is the case in speech recognition where the raw speech input, as represented as amplitude of sound waves over time, is converted into coefficients representing amplitudes *at various frequencies* over time using a fast Fourier transform preprocessing step [Waibel et al., 1987]. This FFT pre-

processing is known to be a useful and widely applicable first step in automatic processing of speech data [O'Shaughnessy, 1987]. Furthermore, algorithms to compute a signal's Fourier transform, while complex, are well understood and have efficient implementations on a variety of computer architectures. Finally, not only is the information contained in the Fourier transform proven useful in previous speech recognition systems, the Fourier transform also has the property that little information in the original signal is lost in the transformation. This insures that important input features are not lost as a result of the FFT.

On the other hand, if appropriate preprocessing algorithms are not known for the task, or if the known algorithms are too complex to be practical, the solution is to give the network the raw input and let it learn to perform its own preprocessing. Such is the case with image processing for autonomous robot guidance. The types of features which are important vary widely with the situation and with the sensor being used. For example, lined highway driving requires attention to different features than single lane dirt road driving. Similarly, video images contain very different features than laser rangefinder images. Furthermore, algorithms that are known to be useful in many image based tasks, such as edge finding and region segmentation, are computationally expensive and eliminate much potentially important information.

As a result of these difficulties with preprocessing images for mobile robot guidance, the philosophy in this work has been to keep the system general by performing a minimal amount of preprocessing. The idea is to let the network learn not only to combine important features to determine the most appropriate response in the current situation, but also to determine what the most important features are and how to detect them in the input image.

2.2.1 Preprocessing Practice

The result of this minimal preprocessing philosophy is an input representation consisting of a two-dimensional retina of units onto which sensor images are projected, as described above. Each unit in the input retina corresponds to a pixel in the sensor image. The sensor images used as input are constrained to a relatively low resolution because of the high computational cost of a large input layer when simulating the network on a serial computer. As a result, the most common retina size used, and the size which can be assumed throughout this dissertation unless otherwise stated, is 30x32 units. Image resolutions of up to 64x60 units have been tried on occasions, particularly in situations where small image features, like

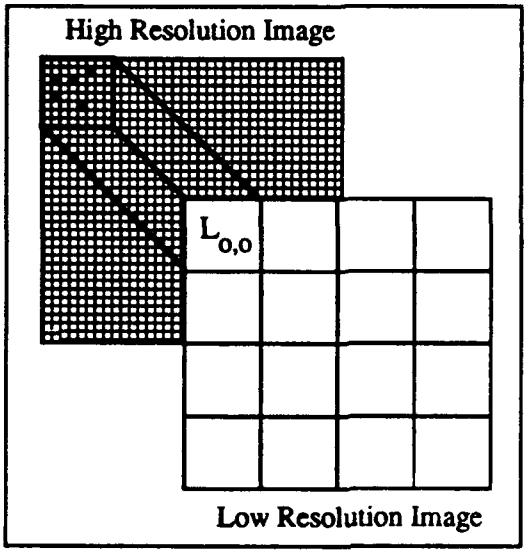


Figure 2.2: To determine the value of a pixel $L_{0,0}$ in the low resolution image, a small percentage of the pixels in the corresponding region of the high resolution image are sampled and averaged together.

the lines painted on the road, are important for accurate driving. However for most domains, including multi-lane highway driving, networks with 30x32 unit input retinas have proven sufficient and at least as capable as networks with higher resolution inputs.

In order to transform the high resolution sensor images into the low resolution images required by the network, a simple subsampling with averaging technique is used. The technique is easiest to understand from example. To determine the value for a unit in the 30x32 unit retina from the pixels in the 480x512 image, a small percentage of the pixels (usually around 3%) in the corresponding 16x16 pixel region of the high resolution image are randomly sampled and averaged together to get a mean value for the region (see Figure 2.2). The resulting low resolution image is then histogrammed and normalized in order to assign activation values to pixel values. More specifically, the darkest 5% of the pixels are assigned the minimum activation level, -1.0, and the brightest 5% of the pixels are assigned the maximum activation level, 1.0. The other 90% of the pixels are assigned activation values proportional to their brightness relative to the extremes. This histogram normalization process ensures that all images the network receives

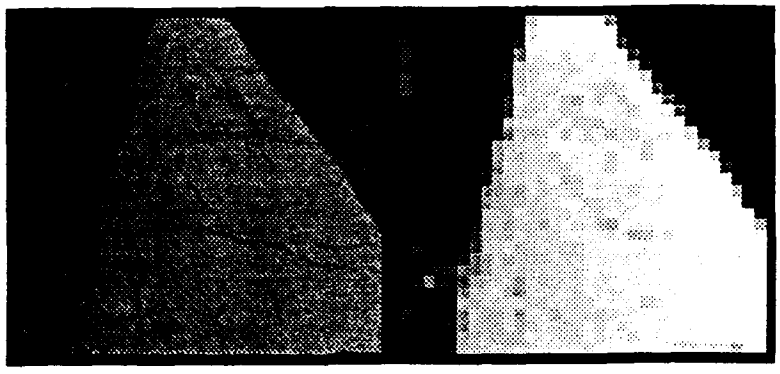


Figure 2.3: A single lane paved road image, and the corresponding reduced and histogram normalized image used as input to the network.

have similar brightness levels, and that the full dynamic range of the input units is employed. In analyzing the histogram normalization technique with respect to the four preprocessing criteria described above, it is a simple, commonly used image processing algorithm known to be useful in a wide variety of situations. It is applicable to all three of the sensor modalities used in this work: video images, laser reflectance images and laser range images. A high resolution black-and-white video image, and the corresponding low resolution image after the histogram normalization are shown in Figure 2.3.

The sensor used most frequently in this work is a color video camera. Color cameras provide three greyscale images or color bands, one each for the red, green and blue components of the image as shown in Figure 2.4. There is therefore an additional preprocessing step required to convert these three bands into a single greyscale image used as input to the network described above. The simplest method of conversion would be to average the pixel values of the three color bands to create a greyscale image, as shown in the left image of Figure 2.5. This is the "intensity" image that would be seen by a monochrome (black and white) camera. This naive averaging approach discards valuable information, which can be used to enhance the contrast between important features in the image and to eliminate the confusing effect of shadows.

The first step towards eliminating shadows is to observe that, at least to first approximation the effect of a shadow on a pixel is to decrease its intensity, but to leave its color (i.e. the relative amounts of red, green, and blue in the pixel) constant. In other words, a pixel corresponding to grass in an image will be darker

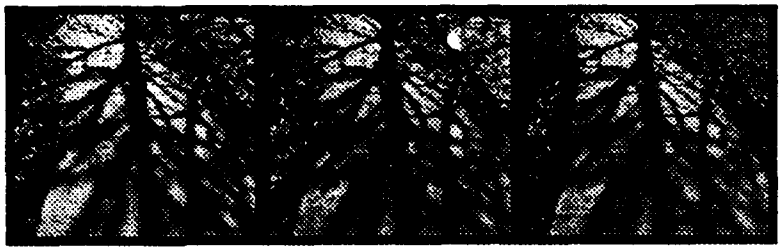


Figure 2.4: The red, green and blue bands of a single color image of a single lane paved road. While it is difficult to distinguish the three color bands when each is represented separately in greyscale, when they are combined into a single image the differences in the color bands become apparent.

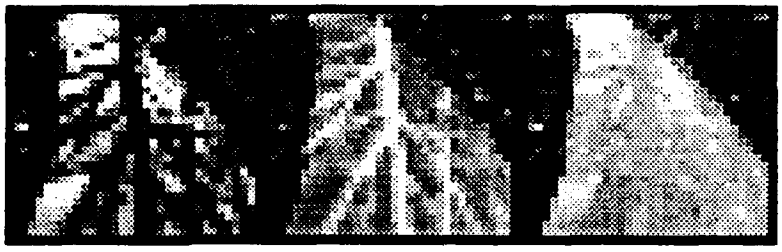


Figure 2.5: Images illustrating three techniques for converting a color image to a greyscale image for input to the network. The left image represents a simple averaging of the three color bands. Notice the strong shadows in this image. The middle image represents the blue band normalized for pixel brightnesses. In this image, the shadows are brighter than the unshadowed regions because of the strong blue component in indirect light coming from the sky and because of camera nonlinearities. The image on the right was made by adding the original blue band and the brightness normalized blue band. Since shadows were dark in the original blue band, and bright in the brightness normalized blue band, they appear to be the nearly the same intensity as the unshadowed regions in this image.

if it is in a shadow, but will still have a much stronger green component than red or blue, just like its unshadowed neighbors. By normalizing pixels for their brightnesses, this relative color constancy can be used to nearly eliminated shadows from outdoor images [Wallace et al., 1985a]. Mathematically, normalizing for brightness can be expressed by the equations,

$$r = \frac{R}{R+G+B}$$

$$g = \frac{G}{R+G+B}$$

$$b = \frac{B}{R+G+B}$$

where R, G and B represent the original red, green and blue components of a pixel and r, g and b represent the red, green and blue components after normalization for brightness. Of course brightness normalization of all three bands results in three images, whereas the network architecture describe above uses only one. Of the three, the normalized blue band best discriminates between important features in the color image, since paved roads have a strong blue component, while other features like dirt, grass, and yellow road center lines are strong in red and/or green but weak in blue. To capitalize on the discrimination power of the normalized blue image, it is included as one component of the input image for the network.

But there are still two problems with the normalized blue image which make it less than ideally suited for use as the sole input to the network. The first problem is illustrated in the center image of Figure 2.5, which represents the reduced resolution normalized blue version of the image from Figure 2.4. In this image the shadows appear *brighter* than the unshadowed regions. These bright shadows result from two factors. First, the illumination of unshadowed regions comes directly from the sun, and therefore has a nearly uniform distribution of colors. Shadowed regions on the other hand are illuminated solely by indirect sources, and in particular by light from the sky. The strong blue component of light from the sky gives pixels in shadowed regions a higher relative blueness than their neighbors in unshadowed regions, making them appear brighter in the normalized blue image. The second factor contributing to the extra blueness of shadowed pixels results from a characteristic of color CCD cameras. Color CCD cameras are more sensitive in low light conditions to blue light than to red or green. As a result, the "percent blueness" of a pixel increases in shadows. In short, in an

attempt to brighten shadows by normalizing the blue band for brightness, we have actually overcompensated for them.

The second problem with using a single normalized color band as the sole input to the network is the fact that relative color isn't *always* a good discriminant between important feature in the image. There are in fact situations, such as driving on a hard-packed dirt road surrounded by soft-packed dirt, where the color of the road and the non-road are nearly identical, and only the brightness differs between the two. To avoid eliminating a potentially crucial discriminating factor by normalizing for brightness, a term representing absolute brightness in the blue band is added to the preprocessed image. The final preprocessed image is derived using the equation

$$p = \frac{B}{255} + \frac{B}{R+G+B}$$

where p represents value for a pixel after preprocessing and 255 is a scale factor to limit the absolute blue component of a pixel to a value between 0.0 and 1.0. The result of this preprocessing can be seen in the right image of Figure 2.5. Notice how the strong shadows in the original image have been almost totally eliminated, and only distinctions based on actual color remain in the image.

2.2.2 Justification of Preprocessing

Having earlier mentioned the goal of maintaining the system's generality by minimizing preprocessing, it is reasonable to ask how I can justify performing this relatively sophisticated image preprocessing. While the preprocessing scheme is relatively simple, it is specific to color video images, and hence violates the generality criterion mentioned above as an important factor in choosing a preprocessing technique. The reason this color image preprocessing is justified is that the network learns to approximate this technique when provided with raw color images.

The experiment went as follows. Instead of performing the preprocessing steps described above, I provided the network with three 30x32 pixel input images, one for each of the three video color bands (See Figure 2.6. Each pixel in each of the three input images had a connection to the corresponding pixel in a 30x32 unit array of hidden units. All the connections from a single color input band to the array of hidden units were constrained to have identical weights. In other words, there were only three distinct weights between the three color input bands

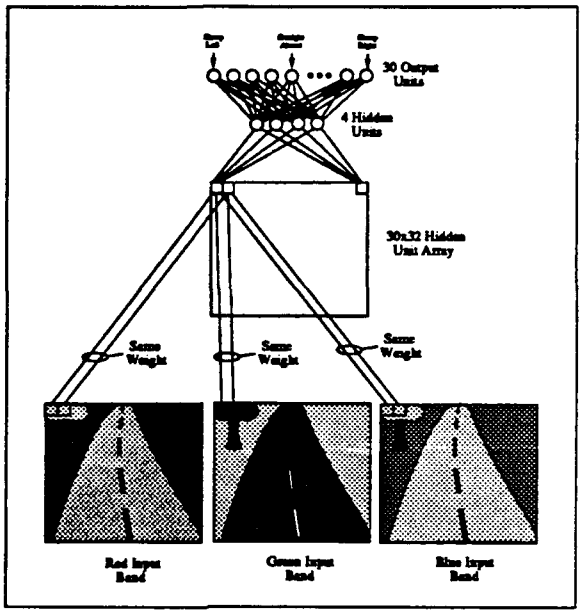


Figure 2.6: The four-layered architecture used to train ALVINN on the three band of the color video signal

and the array of hidden units. It was the job of this first layer of connections to determine appropriate weightings for the three color bands to enhance the contrast of features in the image which would be needed later in the network for determining the direction to steer. Above the array of 30x32 hidden units, the architecture was identical to the standard network, with a layer of 4 hidden units fully connected to a layer of 30 output units.

The network was trained using the training "on-the-fly" techniques described in detail in the next chapter. It was repeatedly presented with road images and the corresponding correct output steering direction. The network's weights were updated using the back-propagation learning rule so as to produce the correct steering direction when presented with a road image. In this case however, the network had to also learn a relative weighting for the three color bands in order to produce an "image" in the first hidden layer with sufficient information for the higher layers to produce the correct steering direction. The color band weighting that the network developed was a surprisingly close approximation to the preprocessing described above. Specifically, to determine a pixel's value in

the first hidden layer, the network learned that the corresponding blue pixel in the input should have a strong positive influence, and the corresponding red and green pixels in the input should have negative influence. As a result, if a pixel in the input image had a lot of blue and little red or green, the corresponding unit in the first hidden layer would be highly active, just like in the normalized blue preprocessing described above. But if the pixel was bright in all three bands, the positive influence from the blue component would be offset by the negative influence of the red and green components. As a result, the corresponding unit in the first hidden layer would have an intermediate activation value, just as in the normalized blue preprocessing described above.

This experiment demonstrates that a network can learn to perform the appropriate preprocessing and subsequently learn to produce the correct steering direction when given only the raw color bands from the video signal. However, the added complexity of the task and the network architecture resulted in a factor of six increase in training time over a three layer network such as Figure 2.1 that was given the normalized blue image as input. Because of this slow down in training, and because of the similar slow down that occurs at runtime when performing the forward pass through the more complex network, it was decided not to force networks to learn the normalized blue preprocessing, but to perform the preprocessing using the algorithm described above and to provide the resulting image as input to the network.

2.3 Output Representation

Another crucial decision in determining the network architecture is the output representation. In the case of mobile robot guidance, the network's output represents a specific motor response along a continuum. For instance, in autonomous driving, the network's output represents the steering direction appropriate for the current situation, ranging continuously from sharp left to sharp right. Similarly, in the foot placement task for the walking robot, the network's output represents the displacement of the robot's foot from a "safe" position to step.

I have investigated three techniques for representing continuous scalar values such as these in connectionist networks. For purposes of concreteness, in the comparison of the three techniques, I will focus on representing the steering direction of a driving network. However the analysis and conclusions hold for representing any bounded continuous value in a neural network.

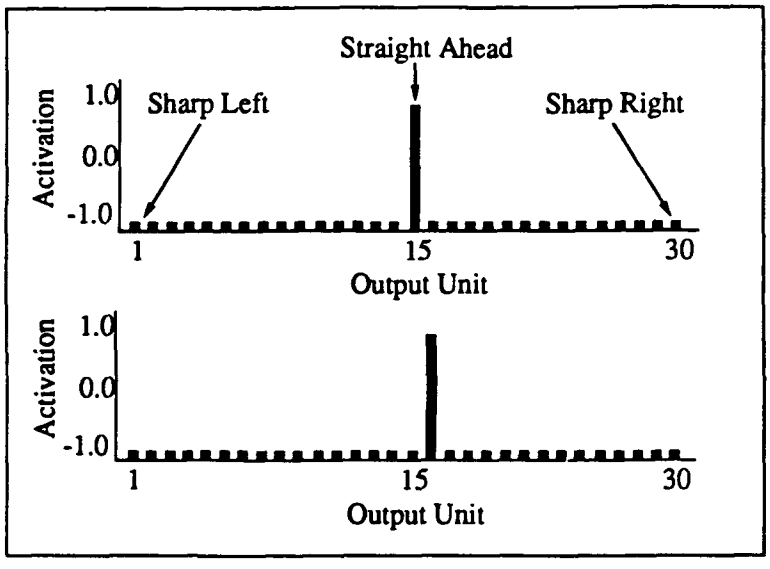


Figure 2.7: The representation of two steering directions using a "1-of-N" encoding. The top graph represents a straight ahead steering direction, since the middle output unit is activated. The bottom graph represents a slight right turn, since an output unit slightly right of center is activated.

2.3.1 1-of-N Output Representation

In the first representation technique, a continuous range of values is discretized and the problem of producing the correct motor response is converted into a 1-of-N classification task. In the case of the driving network, each output unit represents a different, equally spaced steering direction ranging from a sharp left to a sharp right turn. To represent a particular steering direction, the unit representing the closest steering direction to the desired direction would be fully activated with a 1.0 activation level. So for example, the straight ahead steering direction would be represented by fully activating the output unit closest to the center, since it represents the closest steering direction to the desired direction (See the top graph in Figure 2.7). The remainder of the output units would be given a desired activation level of -1.0[1].

There are three problems with the 1-of-N encoding of continuous values. The first problem is that by having each output represent a discrete motor action, the network is limited to a fixed number of responses. With a limited number of responses, the network's ability to precisely control the robot's movement is limited, reducing the system's performance. Of course, it is possible to increase the precision of the network's response by increasing the number of output units, but this in turn aggravates the second problem with the 1-of-N representation.

Another disadvantage of the 1-of-N encoding is that it requires at least one training example for each of the N responses. Without any examples of a training pattern showing a particular output unit active, the network will assume it should always be inactive. There is no possibility of generalizing from training pattern that require "similar" steering directions, because with the 1-of-N encoding there is no notion of similarity between output classes. Instead, each output class is distinct and unrelated to any of the others.

A related problem is that the mapping from input to output units when using the 1-of-N representation is a highly nonlinear function. Consider two road images. The first shows a road centered in the image and heading straight ahead. The desired steering direction for this image would be straight ahead, so the desired output vector would look like the top graph of Figure 2.7. The second image is identical to the first except that the road is shifted slightly to the right. In fact, it is shifted just enough to change the desired steering direction to be closer to

[1]The hyperbolic tangent activation function is used as the activation function for all the units in the network, making their ranges -1.0 to 1.0. The advantages of using a symmetric activation function, instead of the normal sigmoid with a range from 0.0 to 1.0 is discussed in Chapter 6.

the direction represented by the first output unit right of center, instead of the middle output unit, as illustrated in the bottom graph of Figure 2.7. As a result of a very small change in the input image, there is a significant change in the desired output, namely the middle unit has gone from active to inactive and its neighbor to the right has gone from inactive to active. It is difficult for the network to develop a representation that allows changes of just a few pixels in the input image to significantly change in the output vector. This difficulty with highly nonlinear mappings is one of the reasons handwritten character recognition is such a difficult problem. Small differences in the input, such ass the difference between an "F" and an "E", result in a different classification and therefore a very different desired output. In character recognition, this nonlinearity cannot be avoided, since a network must be able to clearly differentiate between an "F" and an "E".

Many of these problems are illustrated in Figure 2.8. It depicts the internal representation developed by a network with the 1-of-N output representation. The network was identical in structure to the network depicted in Figure 2.1, with 30 output units representing 30 different steering directions. The network presented in this example, like those in the examples from the next two sections, was trained by presenting it with 100 images of a real single lane paved road which were digitized as a person drove the Navlab. It was trained to produce the output corresponding to the direction the person was steering in when each of the images was taken. The network was trained until it showed only asymptotic decreases in error on the training set.

In Figure 2.8, the four large rectangles labeled "Input-Hidden1 Weights" through "Input-Hidden4 Weights" represent the weights going from each of the units in the input retina to the four hidden units in the network. Dark squares represent inhibitory connections and light squares represent excitatory connections. The intensity of a square represents the weight's magnitude. The four vectors labeled "Hidden1-Output Weights" through "Hidden4-Output Weights" represent the weight going from each of the four hidden units to the 30 output units. The stripes or bands of similar weights in the input to hidden weights represent edge detectors the network has developed for determining where in the image the road is (For a more detailed analysis of the internal representations developed by ALVINN networks, see Chapter 6). The rectangle in the lower right corner labeled "Input Acts" represents the activations of the units in the input retina on a particular road image. Light squares represent units with positive activation levels, and dark squares represent units with negative activation levels (recall that the range of

activation levels is -1.0 to +1.0). The vectors labeled "Hidden Acts" and "Output Acts" represents the activation levels of the hidden units and the output units on this particular input pattern. The "Target Acts" vector shows the desired activation level of the output units for this input pattern. The input image in this case depicts a road on the right side of the image, so the target output vector has a single unit activated towards the right side, indicating a right turn is desired in this situation.

Notice the sharp differences between weights to neighboring output units from individual hidden units, particularly from hidden units 3 and 4. Also notice in hidden units 3 and 4 the sharply delineated features in the input to hidden unit weights. These abrupt changes are a result of the sharp output differences required by the 1-of-N representation for similar input patterns. Due to the arbitrary and highly specific nature of its internal representation, the network has difficulty generalizing to slightly different input patterns, as will be seen in the performance comparison below.

Also notice that unlike the target vector, in which a single output unit is fully active, the actual output vector has a few units in the vicinity of the correct one slightly activated. This illustrates the networks tendency to treat neighboring output units, representing similar steering directions, similarly. This desirable tendency is exploited in the next two output representations.

2.3.2 Single Graded Unit Output Representation

In many domains, mobile robot guidance being one of them, the output of the network represents a continuous quantity and not simply membership in one of N unrelated classes. A representation which exploits this structure can help eliminate some of the complexity in the mapping from inputs to outputs. One representation that reduces the mapping complexity is a "graded representation", in which the activation level of a single output unit is used to encode the continuous output. So for the output of a driving network, a sharp left steering direction would be represented by an activation level of -1.0, sharp right by an activation level of 1.0, and intermediate steering directions by activation levels between -1.0 and 1.0.

This compact output representation reduces the mapping complexity, since input images which differ only slightly would require only slightly different output responses. But the single graded unit output representation has other disadvantages. With the 1-of-N encoding, the network's support for a response can be measured using the activation levels of the corresponding output unit. The greater a unit's activation level, the greater the network's support for its steering direction.

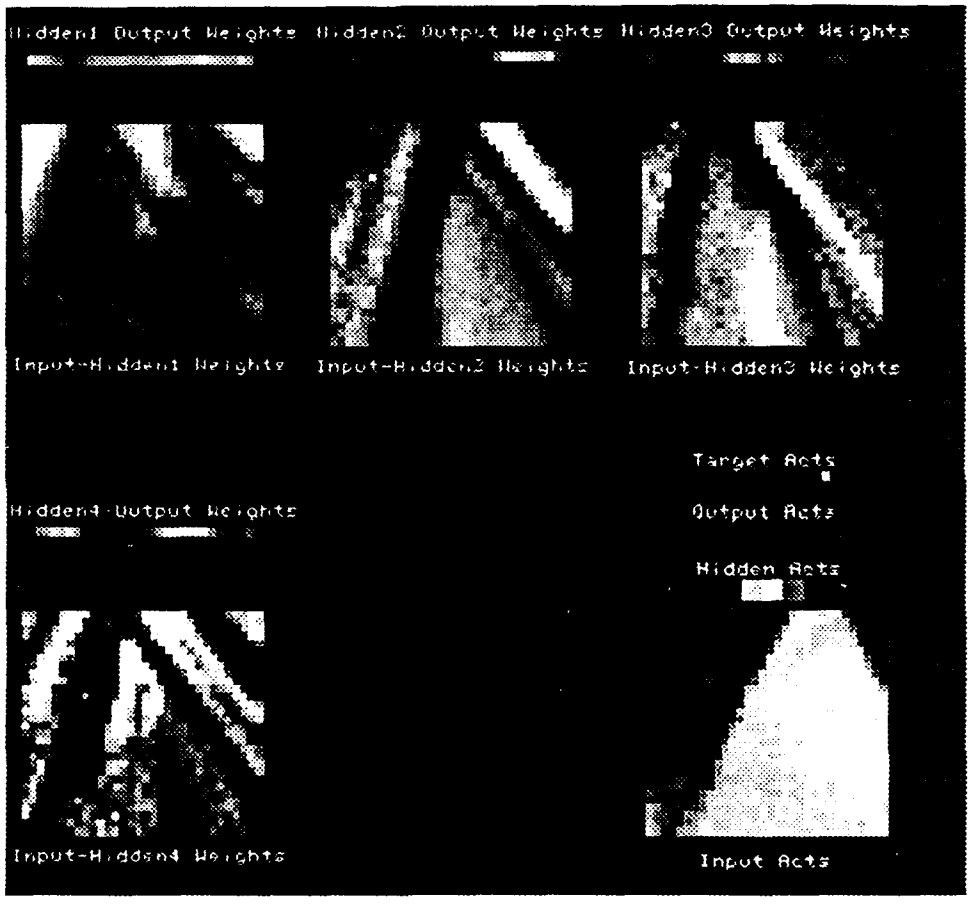


Figure 2.8: The internal representation developed by a network trained on images of a single lane road when using a 1-of-N output representation.

The single graded unit representation has no corresponding "extra parameters" to represent the degree of support for particular responses. Instead, every possible output pattern represents a different response.

This inability to represent support for individual responses results in a number of problems. As will be seen in Chapter 7, the degree to which one and only one steering direction receives support is a good estimate of network reliability. With the single output unit representation, there is no way to quantify support for one steering direction versus another, making this reliability estimation technique impossible. A closely related problem is that with a single output unit it is impossible for the network to discretely select a single response from a number of likely competitors. Suppose for example, certain input features suggest a sharp left turn, while others suggest steering hard to the right. In the 1-of-N representation, the response with the largest support (i.e. output unit with the highest activation level) can be chosen as the direction to steer. But with the single output encoding, input features that support a sharp left turn will result in inhibitory influence on the single output unit. Conversely, input features that suggest turning hard to the right will result in excitatory influence on the output unit. Together these conflicting signals will lead to an intermediate output activation and hence an incorrect, straight ahead steering response.

This inability to represent competing responses in the output has a detrimental influence on the internal representations developed by networks. As will be seen in the analysis of the internal representations of networks in Chapter 6, when using a distributed output representation in which different output units represent different motor responses, hidden units are able to develop a distributed representation. Specifically, hidden units develop into detectors for roads at multiple positions and orientations in the image, and suggest *multiple steering directions as output.* For example, a hidden unit might become active when presented with an image containing a road either on the far left or far right. It would in turn make excitatory connections to output units representing both a sharp left and a sharp right turn. With a single, graded output unit representing steering direction, this type of distributed representation is impossible, since a hidden units can only excite or inhibit the single output unit, suggesting either a right *or* a left turn, but never both.

The detrimental influence the single output unit representation has on the network's internal representation is illustrated in Figure 2.9. This network was trained on the same set of images as in the 1-of-N example above. Instead of 30 output units representing discrete steering directions, this network has a single output whose activation represents the steering direction.

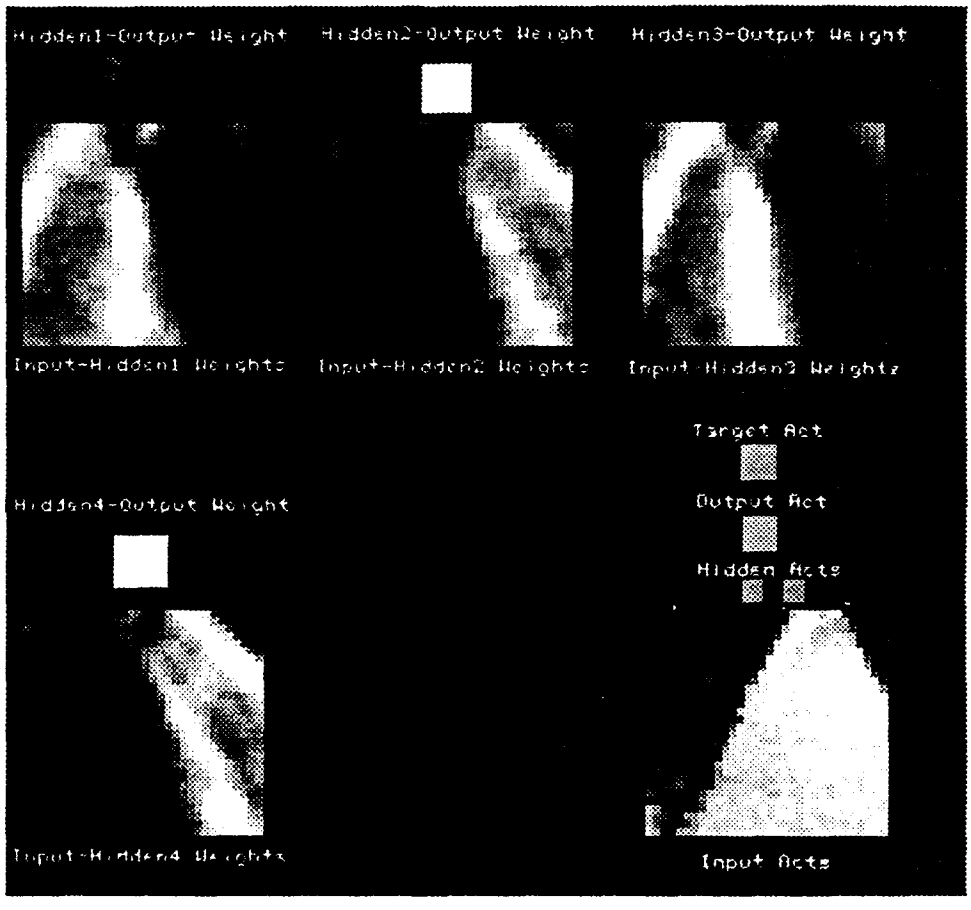


Figure 2.9: The internal representation developed by a network trained on images of a single lane road when using a single graded unit to represent the correct steering direction.

In contrast to the representation developed with the 1-of-N output representation, in this case the hidden units develop very smooth, large scale feature detectors. In fact, it is fairly obvious that each of the hidden units delineates between the road being on the left or the right side of the image. The further left the road appears in the image, the more excited hidden units 1 and 3 become, and the more they inhibit the output unit, indicating a left turn. The further right the road appears in the image, the more excited units 2 and 4 become, and the more they excite the output unit, indicating a right turn. Close observation reveals that hidden units 1 and 3 are nearly identical, as are units 2 and 4. In addition, hidden units 1 and 3 are simply inversions of hidden units 2 and 4, meaning that when units 1 and 3 are active, units 2 and 4 are inactive. Because of the symmetric activation function used, the influence each of the hidden units has on the output unit will be identical for all inputs. In other words, for a road on the right side of the input image, units 2 and 4 will be active and since they have positive connections to the output unit, they will excite it. On the same image, units 1 and 3 will be inactive (i.e. have a negative activation level). Since they make inhibitory connections to the output unit, a negative activation level times a negative weight will result in an excitatory influence. Since all the hidden units have a similar influence on the output unit for all inputs, there is really only a single distinct hidden unit in the network. This collapse of distinct hidden units results from the fact that by only having a single weight to influence the network's output, the hidden units are forced to become general "left-or-right" detectors. This lack of representational freedom severly limits the accuracy with which this network architecture can perform the driving task, as will be seen in the performance comparison between the networks.

2.3.3 Gaussian Output Representation

The question is, how to get the advantages of distributed internal representations and certainty measurements possible with a multi-unit output representation while preserving the simplicity of input-to-output mapping and the arbitrary precision inherent in a single-unit, graded representation. The answer is shown in Figure 2.10. Just as in the 1-of-N representation, there are multiple output units representing different motor responses. But unlike the 1-of-N encoding, to represent a particular motor response, multiple output units are activated in a gaussian pattern, with the gaussian centered on the correct motor response.

In more detail, the following approximation to a gaussian equation is used to

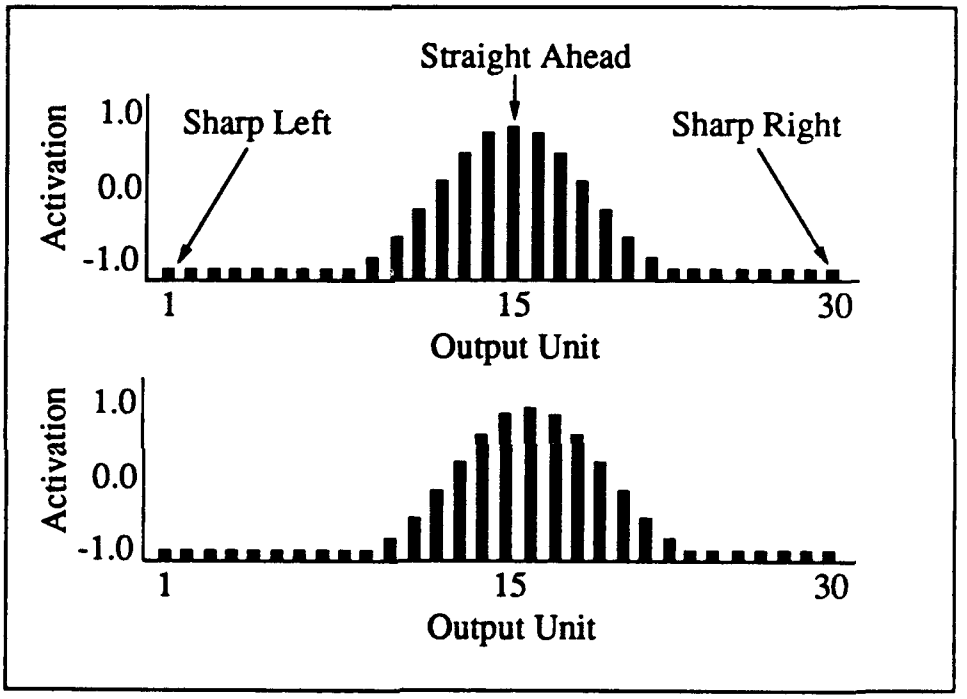


Figure 2.10: The representation of two steering directions using a gaussian output encoding. The top graph represents a straight ahead steering direction, since the gaussian "hill" of activation is centered on the middle output unit. The bottom graph represents a slight right turn, since the "hill" of activation is centered slightly right of the middle unit.

precisely interpolate the correct output activation levels:

$$x_i = e^{-\frac{d_i^2}{10}}$$

where x_i represents the desired activation level for unit i and d_i is the ith unit's distance from the correct steering direction point along the output vector. The constant 10 in the above equation is an empirically determined scale factor that controls the width of the gaussian curve. The above equation corresponds to a normal distribution with a standard deviation $\sigma = \sqrt{10}$.

As an example from the driving network domain, consider the situation in which the correct steering direction falls halfway between the steering directions represented by output units j and $j + 1$. Using the above equation, the desired output activation levels for the units successively farther to the left and the right of the correct steering direction fall off rapidly with the values 0.98, 0.80, 0.54, 0.29, 0.13, 0.04, 0.01, etc.

Just as in the 1-of-N encoding, it is the position of the activation peak along the vector of output units that encodes the motor response. As a result, individual hidden units can support multiple steering directions by making excitatory connections to multiple output units, encouraging a distributed internal representation. Also, since it the position of the peak of activation, and not the activation level itself that determines the motor response, the height and shape of the activation peak can be used to measure the network's certainty (for more details, see Chapters 8 and 9).

But unlike the 1-of-N encoding, a small change in the input will only result in a small change in the output. Consider the two road images mentioned above, with the second depicting the exact same scene as the first, except the road is shifted slightly to the right. The two desired output vectors when using the gaussian output representation are depicted in the top and bottom graphs of Figure 2.10. Notice that because of the large overlap of the two gaussians, none of the output units have drastically changed their desired activation levels. As a result of the smooth mapping from changes in the input to changes in the output, developing a robust and accurate internal representation is much simpler for the network when using a gaussian output representation than when using a 1-of-N output representation. The gaussian desired output vector can be thought of as representing the probability density function for the correct motor response, in which a unit's probability of being correct decreases with distance from the gaussian's center.

Another advantage the gaussian output representation has over the 1-of-N representation is its ability to precisely represent arbitrary motor responses. While the 1-of-N representation can encode only a limited number of motor responses, the gaussian output representation can encode arbitrary responses by centering the gaussian appropriately. For instance, to represent a steering direction which lies midway between the directions represented by two output units, the gaussian "hill" of activation can simply be centered half way between the two output units. In this way, the gaussian output representation allows the network to interpolate between units to produce a precise motor command.

Finally, the gaussian output format has the advantage of biological plausibility. Zipser and Andersen [Zipser & Anderson, 1988] employed a gaussian output format for representing eye position and the spatial location of external visual features in their model of primate posterior parietal neurons. In fact, neurophysiological studies suggest that this type of gaussian representation may be the most common coding format found in the brain [Andersen et al., 1985].

The internal representation developed by a network using the gaussian output representation is shown in Figure 2.11. Like in the 1-of-N example, this network had 30 output units. But unlike the 1-of-N example, the desired output vector contained a group of output units with positive desired activation, as depicted in the "Target Acts" display. The edge and region detectors developed in the input to hidden weights in this network are significantly smoother and broader that those in the 1-of-N network. Also in contrast with the 1-of-N internal representation, the weights from individual hidden units to the output vector appear much smoother. That is, there aren't the sharp differences in weights from a single hidden unit to neighboring output units, as there are in the 1-of-N network. The smoother nature of the internal representation over the 1-of-N network results from the more regular mapping between input and output patterns. In other words, since neighboring output units always have similar values, the influence a hidden unit has on neighboring output units can also be similar.

However the gaussian output representation does not so severly restrict the influence a hidden unit can have on the output so as to force all hidden units to be identical, as in the single graded unit output representation. Individual hidden units can support different, relatively specific steering directions, not just "right" or "left". This can be seen in hidden unit 3. If hidden unit 3 has a positive activation level, it will support steering directions ranging from straight ahead to a moderately sharp left turn (as indicated by the broad band of excitatory weights in the "Hidden3-Output Weights" vector). If unit 3 has a negative activation, it

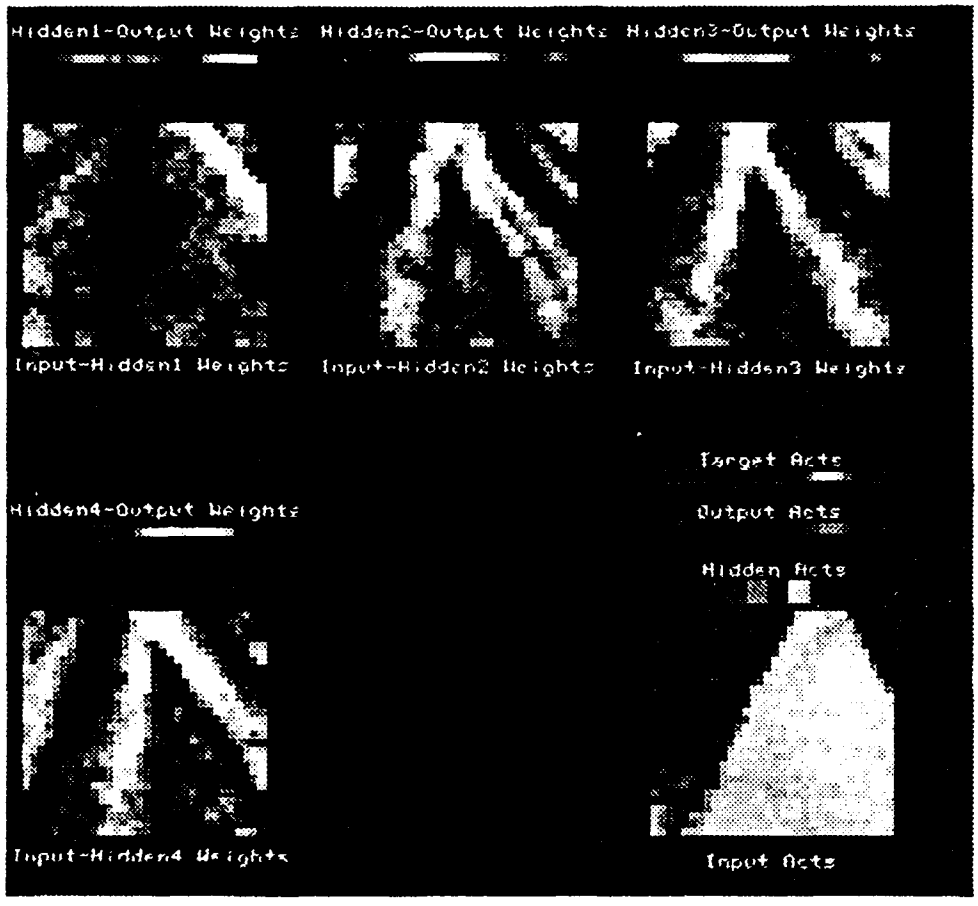


Figure 2.11: The internal representation developed by a network trained on images of a single lane road when using the gaussian output representation for the correct steering direction.

Steering Error of Output Representations

	1-of-N	Single Graded Unit	Gaussian
Error (1/m)	0.00399	0.00474	0.00297

Figure 2.12: A comparison of the steering errors made by networks using each of the three output representations on a test set of 66 images. The height of the bars represent the average absolute difference in curvature between the steering direction indicated by the network and the steering direction indicated by the human driver.

suggests either steering sharply left, or moderately sharp to the right (as indicated by the two short patches of dark, inhibitory weights in the "Hidden3-Output Weights" vector). The performance improvements made possible by the gaussian output representation are illustrated in the next section.

2.3.4 Comparing Output Representations

The steering performance of networks employing the three output representations are depicted in Figure 2.12. The graph shows the average errors made by networks like those described above on a separate test set of 66 road images, taken along a different stretch of the same road the networks were trained on. The errors measures in the graph represent the average absolute difference between the steering direction indicated by the network for an image and the direction the person was steering in when the image was taken, across the 66 test images. Since steering direction is measured in terms of the curvature of the vehicle's current arc, the difference between steering directions is measureed in terms of differences of arc curvatures, which has the units of 1/meters. To eliminate the possible effects of random initial weights, four networks, with different initial weights, were trained for each of the three representations and their results averaged to produce the table.

To convert the output activation levels of the three types of networks into steering directions, the following procedures were used. For the 1-of-N representation, a network's dictated steering direction was interpreted to be the direction represented by the output unit with the maximum activation level. For the single graded unit output representation, the activation level of the output unit was

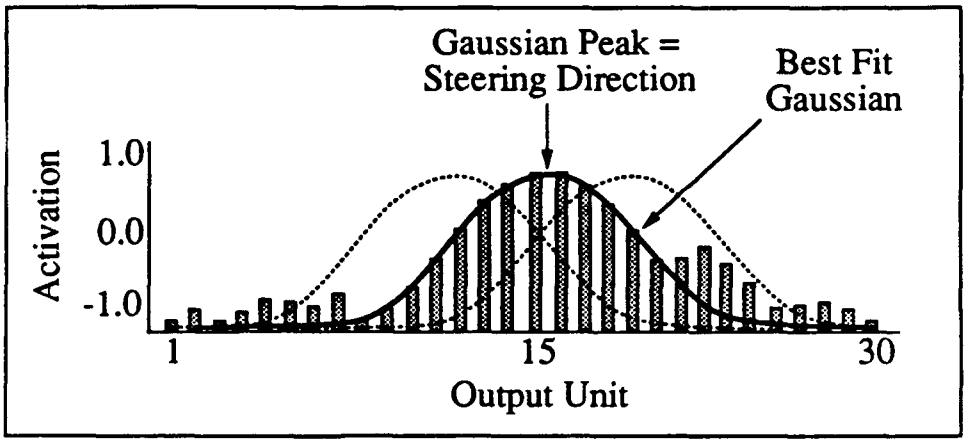


Figure 2.13: To convert the network's output activation levels into a steering direction for the vehicle when using the gaussian output representation, the gaussian of the width specified during training that best fits the output activation levels is determined. The network's steering direction is determined by the position of the best fit gaussian's peak along the output vector.

simply multiplied by a scale factor to determine the network's dictated direction. For the gaussian output representation, the closest gaussian output pattern (with the standard deviation used in training) to the network's actual output pattern was found using a least squares measure of closest fit. The position of the peak of this best fit gaussian along the output vector was used to determine the network's dictated direction (see Figure 2.13).

The average steering error made by networks trained using the 1-of-N output representation was 34% higher than that of networks trained using the gaussian output representation. In addition, networks employing the 1-of-N representation required twice as long on average to reach their optimum performance level than did networks employing the other two representations (80 vs. 160 epochs).

While networks with a single output unit quickly learned the "left vs. right" internal representation depicted in Figure 2.9, the performance of this representation was quite poor. Networks with a single output unit exhibited 59% more steering error on the test set than did networks trained using the gaussian output representation. These results illustrate that the theoretical advantages of the gaussian output representation described above make a significant difference in network performance. More specifically, more regular mapping from inputs to outputs

allows the gaussian representation to generalize better to new situations than the 1-of-N representation. In addition, the ability of the gaussian representation to encode more precise steering directions than the 1-of-N representation makes it more accurate. In comparison with the single graded unit output representation, the gaussian representation encourages more hidden unit diversity, and therefore greater accuracy. As a result of these findings, in the remaider of this dissertation the gaussian output representation is used exclusively.

2.4 Internal Network Structures

The final decision concerning network architecture is determining what comes between the input and the output layers. While the choice of the internal network structure has not been a central focus of this work, I have experimented with a number of different network architectures by varying the number of hidden layers, the size of hidden layers, and the connectivity pattern between layers. While I have not densely sampled the entire space of architectures spanned by these three parameters, I have experimented with networks with 0 to 2 hidden layers, 0-70 hidden units, and connectivity patterns in which hidden units receive connections from either all the units in the previous layer, a randomly selected subset of units in the previous layer, or a spatially contiguous subset of units in the previous layer (i.e., local receptive fields).

My conclusion is that while nearly all of these architectures can do a reasonable job of mapping input images to steering commands, none of them worked significantly better than the network described above with a single layer of four fully-connected hidden units. In more detail, networks with no hidden layer, or a single hidden layer with fewer than four hidden units are less accurate than the chosen network at mapping road images to the appropriate steering direction. This performance shortcoming is particularly pronounced in driving situations where the important image features are spatially small relative to the entire image size. For instance, even a perceptron network can learn to produce reasonably accurate steering directions when presented with single lane road images, in which the feature which determines the correct steering direction is the position of the large region of bright pixels in the image corresponding to the road (See Figure 2.5). However, on multi-lane highway driving, where small painted lines delineate the position of the vehicle relative to the road, the inability of these simpler networks to develop internal representations complex enough to pick out the relevant small

features and at the same time ignore spurious input features results in poorer driving performance.

On the other extreme, networks with additional hidden layers, or more hidden units in a single hidden layer, have not been found to perform significantly better than the network with a single layer of four hidden units. Networks with an additional hidden layer require significantly longer training times to reach the same level of performance as a single hidden layer network. Networks with more than four hidden units take longer to train because of the added network size. They also "waste" much of the additional structure by developing redundant hidden units which respond identically to input images and have identical influence on the output units.

In short, no other network architecture examined has performed significantly better than the simple architecture shown in Figure 2.1. Because of its simplicity and relatively small size, it is used as the basis for the remainder of the experiments and results reported in this dissertation. While slight performance improvements could possibly be achieved through a more detailed study of network architectures, I have focused on improving the system's capabilities using architecturally independent means, including techniques for intelligently training individual networks (see Chapters 3 and 4) and for integrating multiple networks (see Chapters 7, 8 and 9).

Chapter 3

Training Networks "On-The-Fly"

Previous trainable connectionist perception systems have often ignored important aspects of the form and content of available sensor data. Because of the assumed impracticality of training networks to perform realistic high level perception tasks, connectionist researchers have frequently restricted their research to toy problems (e.g., the T-C identification task [Rumelhart, Hinton & Williams, 1986, Jacobs et al., 1990]) or to low level tasks (e.g., edge detection [Koch et al., 1990]). While these restricted domains can provide valuable insight into connectionist architectures and learning algorithms, they frequently ignore the complexities associated with real world problems.

There are exceptions to this trend towards simplified tasks. Notable successes in high level domains such as speech recognition [Waibel et al., 1987], character recognition [LeCun et al., 1989] and face recognition [Cottrell, 1990] have been achieved using real sensor data. However, the results have come only in very controlled environments, after careful preprocessing of the input to segment and label the training exemplars. In addition, these successful connectionist perception systems have ignored the fact that sensor data normally becomes available gradually and not as a monolithic training set. In short, artificial neural networks previously have never been successfully trained using sensor data in real time to perform a real world perception task.

In this chapter, I present techniques which address these shortcomings. The chapter is organized to illustrate the evolution of the training technique used in this work from the traditional monolithic training set approach to the real time training "on-the-fly" technique employed currently.

3.1 Training with Simulated Data

When considering applying artificial neural network techniques to a problem, the first thought of most researchers is to acquire a large and varied set of examples for training purposes. The problem, particularly with complex domains like perception based mobile robot guidance, is how to ensure that the training set is representative of the full range of situations the network might encounter during testing.

My initial attempt at ensuring sufficient diversity in the training set involved generating synthetic images of situations the robot was likely to encounter and using them as training data. The use of synthetic training data was motivated by two factors. First, skepticism concerning the feasibility of using artificial neural networks for mobile robot guidance necessitated a "proof of concept" in simulation before implementing these ideas on a real robot. More importantly from a theoretical standpoint, I believed at the time that the only way to achieve variety in the training set sufficient to ensure that the network learns a general internal representation was to generate the training set synthetically.

To generate synthetic training data for the task of autonomous road following, I developed a program that generated aerial views of simulated stretches of roads and then used a model of the camera to back-project the aerial map into a 2D image of the road ahead. The simulated road image generator used nearly 200 parameters in order to generate a variety of realistic road images. Some of the most important parameters are listed in Figure 3.1.

Figure 3.2 depicts the video images of one real road (left) and one artificial road (right) generated with a single set of values for the parameters from Figure 3.1. At the relatively low resolution employed in the system it is difficult to distinguish between real and simulated roads.

The set of 1200 exemplars used to train the network was created by randomly varying the parameters used by the simulated road generator. Because the artificial road images were created using a detailed model of the scene, the position of the road, and hence the correct direction to steer, was fully specified for each image (the steering model used to map road position to steering direction is discussed in the next section). The network described in the last chapter was then trained using the back-propagation algorithm until the network was making only asymptotic improvements to its steering responses each epoch[1].

[1] Due to the nature of the activation function, eliminating all the network's error on the training

size of video camera retina	3D position of video camera
direction of video camera	field of view of video camera
vehicle position relative to road center	road direction
road curvature	rate of road curvature change
road curve length	road width
rate of road width change	road intensity
left non-road intensity	right non-road intensity
road intensity variability	non-road intensity variability
rate of road intensity change	rate of non-road intensity change
position of image saturation spots	size of image saturation spots
shape of image saturation spots	position of obstacles
size of obstacles	shape of obstacles
intensity of obstacles	shadow size
shadow direction	shadow intensity

Figure 3.1: Parameters varied to generate simulated road images for network training

Figure 3.2: A low resolution video image of a single lane road (left) and an artificial single lane road image created by the road image generator (right).

After training on artificial road images, the network was tested using three techniques. The first test used novel artificial road images. The network correctly dictated a turn curvature within two units of the correct answer approximately 90% of the time on novel artificial road snapshots. A second, more informative test involved "driving" down a simulated stretch of road. Specifically, the artificial road generator had an interactive mode in which the road image scrolled in response to an externally specified speed and direction of travel. After the training described above, the network could drive along a stretch of road created by the road generator at a constant speed for many miles without straying from the simulated road. The best indication of the network's performance came from driving tests on Navlab I, one of CMU's robot vehicles. Specifically, the network could accurately drive Navlab I at a speed of 4 miles per hour along a 400 meter path through a wooded area of the CMU campus under sunny fall conditions.

Despite its apparent success, this training paradigm had serious drawbacks. From a purely practical standpoint, generating the synthetic road scenes was quite time consuming, requiring approximately 6 hours of Sun-4 CPU time. Once the roads scenes were generated, the 30-40 presentations of the 1200 images to train the network required an additional 45 minutes on a 100 MFLOP systolic array supercomputer called the Warp [Pomerleau et al., 1988]. Also, differences between the synthetic road images on which the network was trained and the real situations on which the network was tested often resulted in poor performance in real driving situations. For example, when the network was trained on synthetic road images which were less curved than the test road, it would become confused when presented with a sharp curve during testing. Finally, while relatively effective at training the network to drive under the limited conditions of a single-lane road, it quickly became apparent that extending the synthetic training paradigm to deal with more complex situations such as multi-lane and off-road driving would require prohibitively complex training data generators.

To deal with these problems, I developed a training scheme which involves teaching the network to imitate a human driver under actual driving conditions. The technique, called training "on-the-fly", is described in the next section.

---

set is impossible. It is also quite undesirable, since overlearning on the training set often leads to poor generalization. To prevent this, training was stopped when the network's error improved by less than 0.1% over 5 epochs.

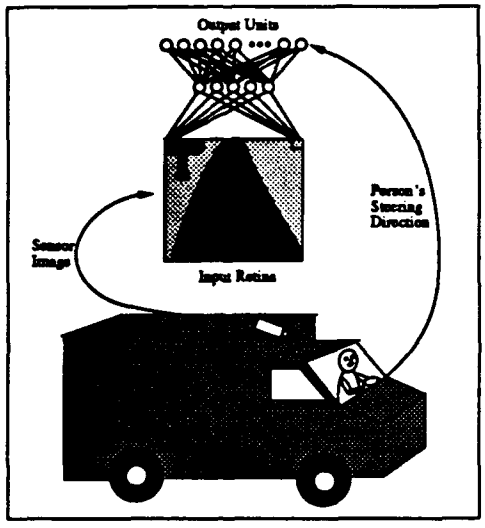


Figure 3.3: Schematic representation of training "on-the-fly". The network is shown images from the onboard sensor and trained to steer in the same direction as the human driver.

3.2 Training "on-the-fly" with Real Data

Autonomous driving has a substantial advantage as a domain for supervised learning: the ready availability of a teaching signal in the form of the human driver's current steering direction. In theory it should be possible to teach a network to imitate a person as they drive using the current sensor image as input and the person's current steering direction as the desired output. This idea of on-line learning is depicted in Figure 3.3.

Training on real images would dramatically reduce the human effort required to develop networks for new situations, by eliminating the need for a hand-programmed training example generator. On-the-fly training should also allow the system to adapt quickly to new situations.

3.2.1 Potential Problems

There are two potential problems associated with training a network using live sensor images as a person drives. First, since the person steers the vehicle down the center of the road during training, the network will never be presented with

situations where it must recover from misalignment errors. When driving for itself, the network may occasionally stray from the road center, so it must be prepared to recover by steering the vehicle back to the middle of the road. The second problem is that naively training the network with only the current video image and steering direction may cause it to overlearn recent inputs. If the person drives the Navlab down a stretch of straight road at the end of training, the network will be presented with a long sequence of similar images. This sustained lack of diversity in the training set will cause the network to "forget" what it had learned about driving on curved roads and instead learn to always steer straight ahead.

Both problems associated with training on-the-fly stem from the fact that back-propagation requires training data which is representative of the full task to be learned. The first approach I contemplated for increasing the training set diversity was to have the driver swerve the vehicle during training. The idea was to teach the network how to recover from mistakes by showing it examples of the person steering the vehicle back to the road center. However this approach was deemed impractical for two reasons. First, training while the driver swerves would require turning learning off while the driver steers the vehicle off the road, and then back on when he swerves back to the road center. Without this ability to toggle the state of learning, the network would incorrectly learn to imitate the person swerving off the road as well as back on. While possible, turning learning on and off would require substantial manual input during the training process, which I wanted to avoid. The second problem with training by swerving is that it would require swerving in many circumstances to enable the network to learn a general representation. This would be time consuming, and also dangerous when training in traffic.

3.2.2 Solution - Transform the Sensor Image

To achieve sufficient diversity of real sensor images in the training set, without the problems associated with training by swerving, I have developed a technique for transforming sensor images to create additional training exemplars. Instead of presenting the network with only the current sensor image and steering direction, each sensor image is shifted and rotated in software to create additional images in which the vehicle appears to be situated differently relative to the environment (See Figure 3.4). The sensor's position and orientation relative to the ground plane are known, so precise transformations can be achieved using perspective geometry.

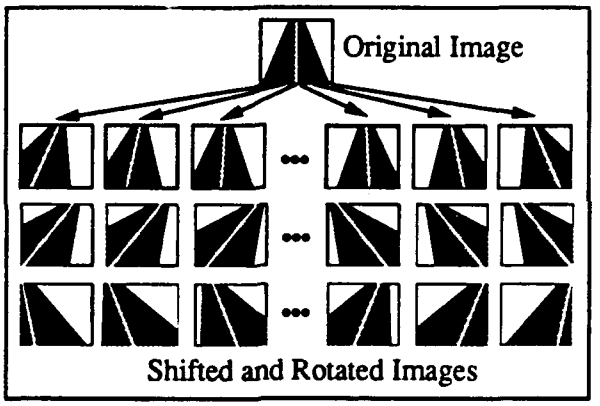


Figure 3.4: The single original video image is shifted and rotated to create multiple training exemplars in which the vehicle appears to be at different locations relative to the road.

The image transformation is performed by first determining the area of the ground plane which is visible in the original image, and the area that should be visible in the transformed image. These areas form two overlapping trapezoids as illustrated by the aerial view in Figure 3.5. To determine the appropriate value for a pixel in the transformed image, that pixel is projected onto the ground plane, and then back-projected into the original image. The value of the corresponding pixel in the original image is used as the value for the pixel in the transformed image. One important thing to realize is that the pixel-to-pixel mapping which implements a particular transformation is constant. In other words, assuming a planar world, the pixels which need to be sampled in the original image in order to achieve a specific shift and translation in the transformed image always remain the same. In the actual implementation of the image transformation technique, ALVINN takes advantage of this fact by precomputing the pixels that need to be sampled in order to perform the desired shifts and translations. As a result, transforming the original image to change the apparent position of the vehicle simply involves changing the pixel sampling pattern during the image reduction phase of preprocessing. Therefore, creating a low resolution transformed image takes no more time than it takes to reduce the image resolution as described in Chapter 2.

Obviously the environment is not always flat. But the elevation changes due to

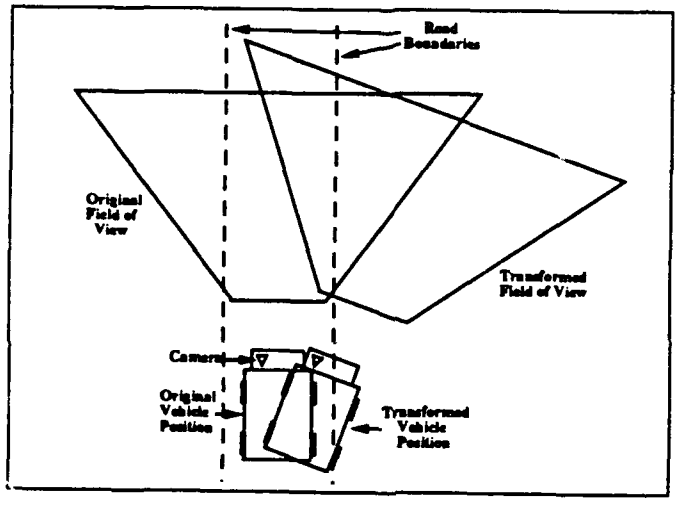


Figure 3.5: An aerial view of the vehicle at two different positions, with the corresponding sensor fields of view. To simulate the image transformation that would result from such a change in position and orientation of the vehicle, the overlap between the two field of view trapezoids is computed and used to direct resampling of the original image.

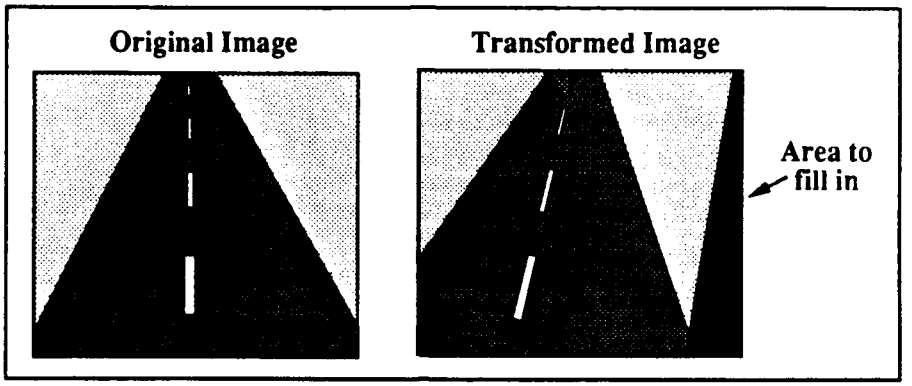


Figure 3.6: A schematic example of an original image, and a transformed image in which the vehicle appears one meter to the right of its initial position. The white region on the right of the transformed image corresponds to an unseen area in the original image. These pixels must be extrapolated from the information in the original image. Notice the number of pixels per row whose value needs to be extrapolated is greater near the bottom of the image than at the top. This is because the one meter of unknown ground plane to the right of the visible boundary in the original image covers more pixels at the bottom than at the top.

hills or dips in the road are small enough so as not to significantly violate the planar world assumption. For more abrupt elevation changes caused by obstacles in the vehicle's path, a more sophisticated image transformation scheme (described in Chapter 6) is required.

Extrapolating Missing Pixels

The less than complete overlap between the trapezoids of Figure 3.5 illustrates the need for one additional step in the image transformation scheme. The extra step involves determining values for pixels which have no corresponding pixel in the original image. Consider the transformation illustrated in Figure 3.6. To make it appear that the vehicle is situated one meter to the right of its position in the original image requires not only shifting pixels in the original image to the left, but also filling in the unknown pixels along the right edge. I have experimented with two techniques for extrapolating values for these unknown pixels (See Figure 3.7).

In the first technique, to determine the value for a pixel that projects to the

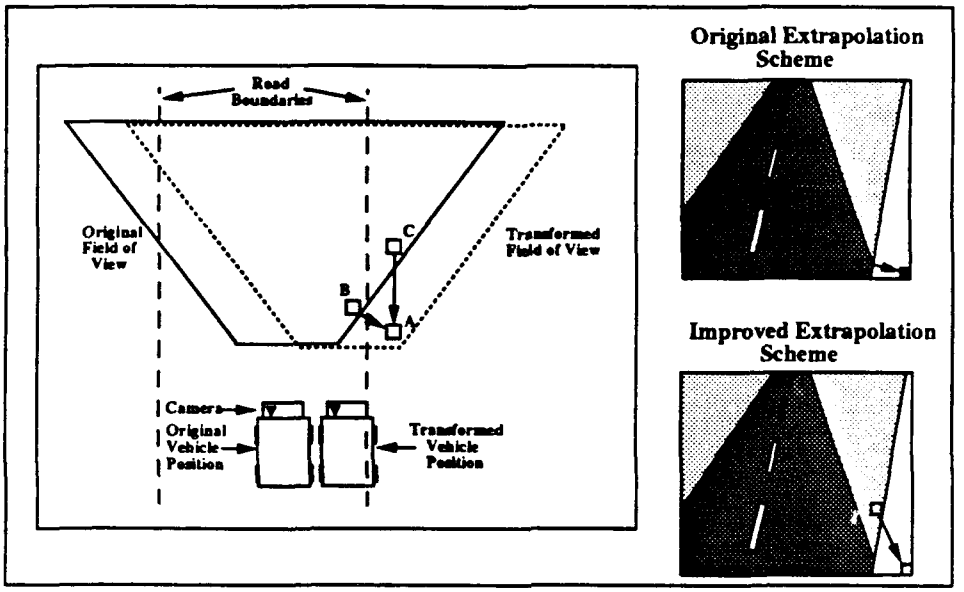


Figure 3.7: An aerial view (left) and image based view (right) of the two techniques used to extrapolate the values for unknown pixels. See text for explanation.

ground plane at point A in the transformed image, the closest ground plane point in the original viewing trapezoid (point B) is found. This point is then back-projected into the original image to find the appropriate pixel to sample. The image in the top right shows the sampling performed to fill in the missing pixel using this extrapolation scheme. The problem with this technique is that it results in the "smearing" of the image approximately along rows of the image, as illustrated in the middle image of Figure 3.8. In this figure, the leftmost image represents an actual reduced resolution image of a two-lane road coming from the camera. Notice the painted lines delineating the center and right boundaries of the lane. The middle image shows the original image transformed to make it appear that the vehicle is one meter to the right of its original position using the extrapolation technique described above. The line down the right side of the road can be seen smearing to the right where it intersects the border of the original image. Because the length of this smear is highly correlated with the correct steering direction, the network learns to depend on the size of this smear to predict the correct steering direction. When driving on its own however, this lateral smearing of features is not present, so the network performs poorly.

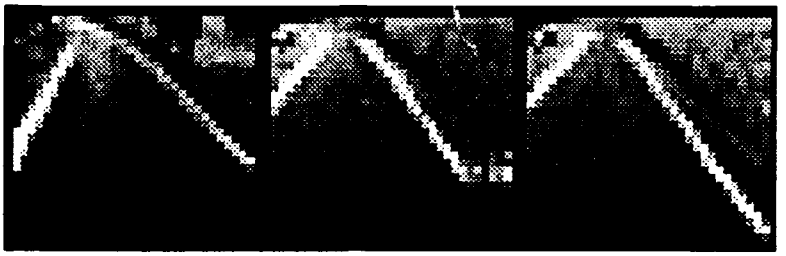


Figure 3.8: Three reduced resolution images of a two-lane road with lines painted down the middle and right side. The left image is the original coming directly from the camera. The middle image was created by shifting the original image to make it appear the vehicle was situated one meter to the right of its original position using the first extrapolation technique described in the text. The right image shows the same shift of the original image, but using the more realistic extrapolation technique.

To eliminate this artifact of the transformation process, I implemented a more realistic extrapolation technique which relies on the fact that interesting features (like road edges and painted lane markers) normally run parallel to the road, and hence parallel to the vehicle's current direction. With this assumption, to extrapolate a value for the unknown pixel A in Figure 3.7, the appropriate ground plane point to sample from the original image's viewing trapezoid is not the closest point (point B), but the nearest point in the original image's viewing trapezoid along the line that runs through point A and is parallel to the vehicle's original heading (point C).

The effect this improved extrapolation technique has on the transformed image can be seen schematically in the bottom image on the right of Figure 3.7. This technique results in extrapolation along the line connecting a missing pixel to the vanishing point, as illustrated in the lower right image. The realism advantage this extrapolation technique has over the previous scheme can be seen by comparing the image on the right of Figure 3.8 with the middle image. The line delineating the right side of the lane, which was unrealistically smeared using the previous method, is smoothly extended in the image on the right, which was created by shifting the original image by the same amount as in the middle image, but using the improved extrapolation method.

The improved transformation scheme certainly makes the transformed images look more realistic, but to test whether it improves the network's driving per-

formance, I did the following experiment. I first collected actual two-lane road images like the one shown on the left side of Figure 3.8 along with the direction the driver was steering when the images were taken. I then trained two networks on this set of images. The first network was trained using the naive transformation scheme and the second using the improved transformation scheme. The magnitude of the shifts and rotations, along with the buffering scheme used in the training process are described in detail below. The networks were then tested on a disjoint set of real two-lane road images, and the steering direction dictated by the networks was compared with the person's steering direction on those images. The network trained using the more realistic transformation scheme exhibited 37% less steering error on the 100 test images than the network trained using the naive transformation scheme. In more detail, the amount of steering error a network produces is measured throughout this dissertation as the distance, in number of units (i.e. neurons), between the peak of the network's "hill" of activation in the output vector and the "correct" position, in this case the direction the person was actually steering in. This steering error measurement is illustrated in Figure 3.9. In this case, the network trained with the naive transformation technique had an average steering error across the 100 test images of 3.5 units, while the network trained with the realistic transformations technique had an average steering error of only 2.2 units.

3.2.3 Transforming the Steering Direction

As important as the technique for transforming the input images is the method used to determine the correct steering direction for each of the transformed images. The correct steering direction as dictated by the driver for the original image must be altered for each of the transformed images to account for the altered vehicle placement. This is done using a simple model called pure pursuit steering [Wallace et al., 1985b]. In the pure pursuit model, the "correct" steering direction is the one that will bring the vehicle to a desired location (usually the center of the road) a fixed distance ahead. The idea underlying pure pursuit steering is illustrated in Figure 3.10. With the vehicle at position A, driving for a predetermined distance along the person's current steering arc would bring the vehicle to a "target" point T, which is assumed to be in the center of the road.

After transforming the image with a horizontal shift s and rotation θ to make it appear that the vehicle is at point B, the appropriate steering direction according to the pure pursuit model would also bring the vehicle to the target point T.

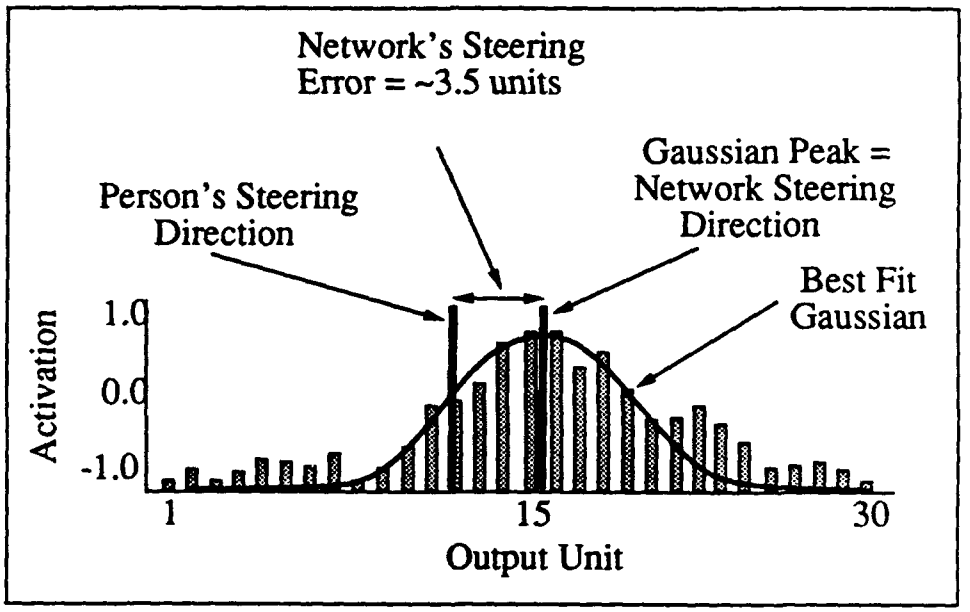


Figure 3.9: To calculate a network's steering error the best fit gaussian is found to the network's output activation profile, as described in Chapter 2. The distance between the peak of the best fit gaussian and the position in the output vector representing the reference steering direction (in this case the person's steering direction) is calculated. This distance, measured in units or neurons between the two positions, is defined to be the network's steering error.

Mathematically, the formula to compute the radius of the steering arc that will take the vehicle from point B to point T is

$$r = \frac{l^2 + d^2}{2d}$$

where r is the steering radius l is the lookahead distance and d is the distance from point T the vehicle would end up at if driven straight ahead from point B for distance l. The displacement d can be determined using the following formula:

$$d = \cos\theta \cdot (d_p + s + l\tan\theta)$$

where d_p is the distance from point T the vehicle would end up if it drove straight ahead from point A for the lookahead distance l, s is the horizontal distance from point A to B, and θ is the vehicle rotation from point A to B. The quantity d_p can be calculated using the following equation:

$$d_p = r_p - \sqrt{r_p^2 - l^2}$$

where r_p is the radius of the arc the person was steering along when the image was taken.

The only remaining unspecified parameter in the pure pursuit model is l, the distance ahead of the vehicle to select a point to steer towards. Empirically, I have found that over the speed range of 5 to 55 mph, accurate and stable vehicle control can be achieved using the following rule: look ahead the distance the vehicle will travel in 2-3 seconds.

Interestingly, with this empirically determined rule for choosing the lookahead distance, the pure pursuit model of steering is a fairly good approximation to how people actually steer. Reid, Solowka and Billing [Reid, Solowka & Billing, 1981] found that at 50km/h, human subjects responded to a 1m lateral vehicle displacement with a steering radius ranging from 511m to 1194m. With a lookahead equal to the distance the vehicle will travel in 2.3 seconds, the pure pursuit model dictates a steering radius of 594m, within the range of human responses. Similarly, human subjects reacted to a 1 degree heading error relative to the current road direction with a steering radius ranging from 719m to 970m. Again using the 2.3 second travel distance for lookahead, the pure pursuit steering model's dictated radius of 945m falls within the range of human responses.

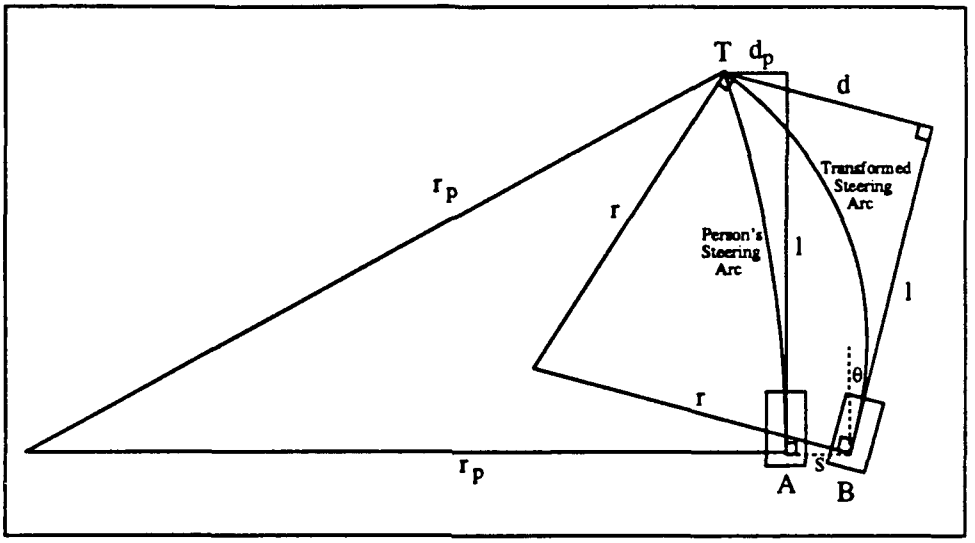


Figure 3.10: Illustration of the "pure pursuit" model of steering. See text for explanation.

Like the image transformation scheme, the steering direction transformation technique uses a simple model to determine how a change in the vehicle's position and/or orientation would affect the situation. In the image transformation scheme, a planar world hypothesis and rules of perspective projection are used to determine how changing the vehicle's position and/or orientation would affect the sensor image of the scene ahead of the vehicle. In the steering direction transformation technique, a model of how people drive is used to determine how a particular vehicle transformation should alter the correct steering direction. In both cases, the transformation techniques are independent of the driving situation. The person could be driving on a single lane dirt road or a multi lane highway: the transformation techniques would be the same.

Anthropomorphically speaking, transforming the sensor image to create more training images is equivalent to telling the network "I don't know what features in the image are important for determining the correct direction to steer, but whatever they are, here are some other positions and orientations you may see them in". Similarly, the technique for transforming the steering direction for each of these new training images is equivalent to telling the network "whatever the important features are, if you see them in this new position and orientation, here is how your

response should change". Because it does not rely on a strong model of what important image features look like, but instead acquires this knowledge through training, the system is able to drive in a wide variety of circumstances, as will be seen in the Chapter 5.

These weak models are enough to solve the two problems associated with training in real time on sensor data. Specifically, using transformed training patterns allows the network to learn how to recover from driving mistakes that it would not otherwise encounter as the person drives. Also, overtraining on repetitive images is less of a problem, since the transformed training exemplars maintain variety in the training set.

3.2.4 Adding Diversity Through Buffering

As additional insurance against the effects of repetitive exemplars, the training set diversity is further increased by maintaining a buffer of previously encountered training patterns. When new training patterns are acquired through digitizing and transforming the current sensor image, they are added to the buffer, while older patterns are removed. I have experimented with four techniques for determining which patterns to replace. The first is to replace oldest patterns first. Using this scheme, the training pattern buffer represents a history of the driving situations encountered recently. But if the driving situation remains unchanged for a period of time, such as during an extended right turn, the buffer will loose its diversity and become filled with right turn patterns. The second technique is to randomly choose old patterns to be replaced by new ones. Using this technique, the laws of probability help ensure somewhat more diversity than the oldest pattern replacement scheme, but the buffer will still become biased during monotonous stretches.

The next solution I developed to encourage diversity in the training was to replace those patterns on which the network was making the lowest error, as measured by the sum squared difference between the network's output and the desired output. The idea was to eliminate the patterns the network was performing best on, and leave in the training set those images the network was still having trouble with. The problem with this technique results from the fact that the human driver doesn't *always* steer in the correct direction. Occasionally he may have a lapse of attention for a moment and steer in an incorrect direction for the current situation. If a training exemplar was collected during this momentary lapse, under this replacement scheme it will remain there in the training buffer for a long time, since the network will have trouble outputting a steering response to match the

person's incorrect steering command. In fact, using this replacement technique, the only way the pattern would be removed from the training set would be if the network learned to duplicate the incorrect steering response, obviously not a desired outcome. I considered replacing both the patterns with the lowest error *and* the patterns with the highest error, but decided against it since high network error on a pattern might also result on novel input image with a correct response associated with it. A better method to eliminate this problem is to add a random replacement probability to all patterns in the training buffer. This ensured that even if the network never learns to produce the same steering response as the person on an image, that image will eventually be eliminated from the training set.

While this augmented lowest-error-replacement technique did a reasonable job of maintaining diversity in the training set, I found a more straightforward way of accomplishing the same result. To make sure the buffer of training patterns does not become biased towards one steering direction, I add a constraint to ensure that the mean steering direction of all the patterns in the buffer is as close to straight ahead as possible. When choosing the pattern to replace, I select the pattern whose replacement will bring the average steering direction closest to straight. For instance, if the training pattern buffer had more right turns than left, and a left turn image was just collected, one of the right turn images in the buffer would be chosen for replacement to move the average steering direction towards straight ahead. If the buffer already had a straight ahead average steering direction, then an old pattern requiring a similar steering direction the new one would be replaced in order to maintain the buffer's unbiased nature. By actively compensating for steering bias in the training buffer, the network never learns to consistently favor one steering direction over another. This active bias compensation is a way to build into the network a known constraint about steering: in the long run right and left turns occur with equal frequency.

The final details required to specify the training on-the-fly process are the number and magnitude of transformations to use for training the network. The following quantities have been determined empirically to provide sufficient diversity to allow networks to learn to drive in a wide variety of situations (see Chapter 5 for details). The original sensor image is shifted and rotated 14 times using the technique describe above to create 14 training exemplars. The size of the shift for each of the transformed exemplars is chosen randomly from the range -0.6 to +0.6 meters, and the amount of rotation is chosen from the range -6.0 to +6.0 degrees. But before the randomly selected shift and rotation is performed on the original image, the steering direction that would be appropriate for the resulting

transformed image is computed using the formulas given above. If the resulting steering direction is sharper than the sharpest turn representable by the network's output (usually a turn with a 30m radius), then the transformation is disallowed and a new shift distance and rotation magnitude are randomly chosen. By eliminating extreme and unlikely conditions from the training set, such as when the road is shifted far to the right and vehicle is heading sharply to the left, the network is able to devote more of its representation capability to handling plausible scenarios. For more details on the network's internal representation, see Chapter 6.

The 14 transformed training patterns, along with the single pattern created by pairing the current sensor image with the current steering direction, are inserted into the buffer of 200 patterns using the replacement strategy described above. After this replacement process, one forward and one backward pass of the back-propagation algorithm is performed on the 200 exemplars to update the network's weights. The entire process is then repeated. Each cycle requires approximately 2.5 seconds on the three Sun Sparcstations onboard the vehicle. One of the Sparcstation performs the sensor image acquisition and preprocessing, the second implements the neural network simulation, and the third takes care of communicating with the vehicle controller and displaying system parameters for the human observer. The network requires approximately 100 iterations through this digitize-replace-train cycle to learn to drive in the domains that have been tested. At 2.5 seconds per cycle, training takes approximately four minutes of human driving over a sample stretch of road. During the training phase, the person drives at approximately the speed at which the network will be tested, which ranges from 5 to 55 miles per hour.

3.3 Performance Improvement Using Transformations

The performance advantage this technique of transforming and buffering training patterns offers over the more naive methods of training on real sensor data is illustrated in Figure 3.11. This graph shows the vehicle's displacement from the road center measured as three different networks drove at 4 mph over a 100 meter section of a single lane paved bike path which included a straight stretch and turns to the left and right. The three networks were trained over a 150 meter stretch of the path which was disjoint from the test section and which ended in an extended

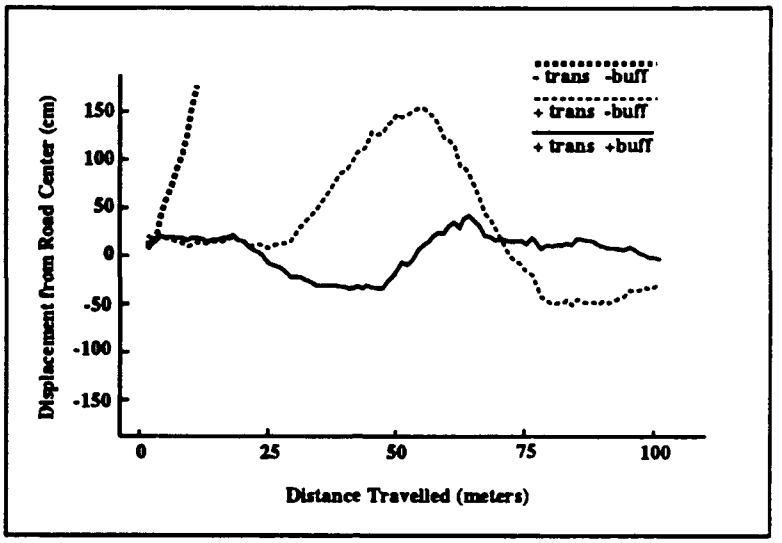


Figure 3.11: Vehicle displacement from the road center as the Navlab was driven by networks trained using three different techniques.

right turn.

The first network, labeled "-trans -buff", was trained using just the images coming from the video camera. That is, during the training phase, an image was digitized from the camera and fed into the network. One forward and backward pass of back-propagation was performed on that training exemplar, and then the process was repeated. The second network, labeled "+trans -buff", was trained using the following technique. An image was digitized from the camera and then transformed 14 times to create 15 new training patterns as described above. A forward and backwards pass of back-propagation was then performed on each of these 15 training patterns and then the process was repeated. The third network, labeled "+trans +buff" was trained using the same transformation scheme as the second network, but with the addition of the image buffering technique desc.. ,ed above to prevent overtraining on recent images.

Note that all three networks were presented with the same number of images. The transformation and buffering schemes did not influence the quantity of data the networks were trained on, only its distribution. The "-trans -buff" network was trained on closely spaced actual video images. The "+trans -buff" network was presented with 15 times fewer actual images, but its training set also contained 14

transformed images for every "real" one. The "+trans +buff" network collected even fewer live images, since it performed a forward and backward pass through its buffer of 200 patterns before digitizing a new one.

The accuracy of each of the three networks was determined by manually measuring the vehicle's lateral displacement relative to the road center as each network drove, using the technique described in Chapter 5. The network trained on only the current video image quickly drove off the right side of the road, as indicated by its rapidly increasing displacement from the road center. The problem was that the network overlearned the right turn at the end of training and became biased towards turning right. Because of the increased diversity provided by the image transformation scheme, the second network performed much better than the first. It was able to follow the entire test stretch of road. However it still had a tendency to steer too much to the right, as illustrated in the graph by the vehicle's positive displacement over most of the test run. In fact, the mean position of the vehicle was 28.9cm right of the road center during the test. The variability of the errors made by this network was also quite large, as illustrated by the wide range of vehicle displacement in the "+trans -buff" graph. Quantitatively, the standard deviation of this network's displacement was 62.7cm.

The addition of buffering previously encountered training patterns eliminated the right bias in the third network, and also greatly reduced the magnitude of the vehicle's displacement from the road center, as evidenced by the "+trans +buff" graph. While the third network drove, the average position of the vehicle was 2.7cm right of center, with a standard deviation of only 14.8cm. This represents a 423% improvement in driving accuracy.

3.4 Discussion

Because of the variety of driving situations and environmental conditions an autonomous navigation system is likely to encounter, training such a system involves special difficulties. Generating realistic artificial training data proved impractical for all but the simplest driving situations. But training on real sensor data is accompanied by its own set of challenges. Prominent among these difficulties is the need to maintain sufficient variety in the training set to ensure that the network develops a sufficiently general representation of the task. Two characteristics of real sensor data collected as a person drives which make training set variety difficult to maintain are temporal correlations and the limited range of situations encountered.

Extended intervals of nearly identical sensor input can bias a network's internal representation and reduce later driving accuracy. The human trainer's high degree of driving accuracy severely restricts the variety of situations covered by the raw sensor data.

The techniques for training "on-the-fly" described in this chapter solve these difficulties. The key idea underlying training on-the-fly is that a model of the process generating the live training data can be used to augment the training set with additional realistic patterns. By modeling both the imaging process and the steering behavior of the human driver, training on-the-fly generates patterns with sufficient variety to allow networks to learn a robust representation of individual driving domains.

In many ways, training on-the-fly blurs the distinction between training on simulated patterns and training on real sensor input. By transforming live sensor images to simulate a new vehicle position relative to the environment, training on-the-fly preserves real input feature characteristics while at the same time generating novel driving situations. The generalization improvements made possible by this type of task modeling will be further demonstrated in the next chapter on adding structured noise to the training set.

Chapter 4

Training Networks With Structured Noise

In the last chapter, I presented a technique for achieving greater training set diversity by geometrically transforming the original training patterns. This technique provides sufficient variety to allow a network to drive reliably as long as the important image features, such as lane markers and road boundaries, remain consistent in appearance between training and testing. However there are situations which give networks trained using the on-the-fly technique trouble. If the appearance of the road changes drastically during testing, say by growing from one lane to two, then no network trained on images of the original road will be capable of driving on the new road. To cope with changing road types, I have developed the multi-network arbitration techniques described in Chapters 7, 8 and 9. These techniques allow the system to switch from a network trained for one situation to a network trained for another, as appropriate.

4.1 Transitory Feature Problem

However there are more subtle and transitory changes in the driving situation which should not require changing networks, but which have proved troublesome for networks trained using the technique described previously. Two examples of temporary but problematic features are illustrated in the real video images of Figure 4.1. The left image shows a typical scene, similar to the ones used for training the network. It has features including the lines down the left, right and

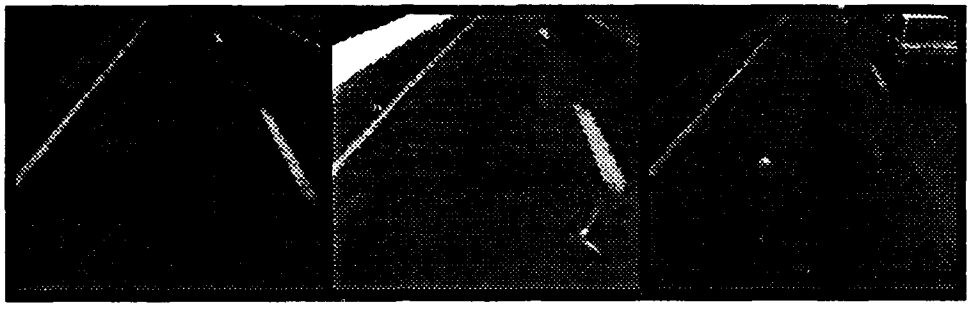


Figure 4.1: Three video images of a multi-lane highway. The image on the left is a typical scene, with visible features including the lane markers and a patch of grass from the median visible in the upper left corner. The center image shows the vehicle going over a bridge. Notice that the grass in the median has disappeared and a jersey barrier has taken its place. The image on the right shows a different transitory image feature, namely a car as it is being passed.

center of the road, and a patch of grass from the median in the upper left corner. The other two images illustrate deviations from this typical scene. The center image shows a jersey barrier instead of a patch of grass in the upper left corner as the vehicle drives over a bridge. The right image shows the vehicle passing another car.

The reason this type of transitory disturbance causes trouble is that each network is trained over a relatively short stretch of road (< 2 miles). As a result, during training the network is not exposed to all the possible driving situations it might encounter when driving autonomously. In particular, since situations like the two illustrated in Figure 4.1 are relatively rare and limited in duration, even if the network sees them during training, it doesn't learn enough from its brief exposure to handle them appropriately. As a result, while a network trained with the technique described in the last chapter is capable of reliably driving in situations closely resembling those it was trained on, when it encounters slightly novel situations like the ones illustrated above, it frequently steers incorrectly.

The influence spurious image features can have on driving performance can be seen by comparing Figures 4.2 and 4.3. Figure 4.2 illustrates a typical reduced resolution multi-lane highway image like the ones ALVINN was trained on. The dark triangle in the upper left is the green grass on the left side of the road. The

thin dark line parallel to it is a yellow line painted down the left side of the road[1]. The faint light dashes towards to upper right corner are part of the dashed white lines marking the lanes. Notice that ALVINN's output response is nearly identical in position to the target, indicating ALVINN is steering in the correct direction. Figure 4.3 is similar to Figure 4.2, except in this input image, there is a guardrail on the left side of the road, which appears as the bright triangular region in the upper left corner. Notice ALVINN's steering direction is greatly disturbed by this relatively minor change to the image.

The reason for this large disturbance is illustrated in the weight diagram of Figure 4.4. In this figure, each of the rectangles labeled "Input-Hidden1 Weights" through "Input-Hidden4 Weights" represents the weights projecting from the input retina to one of the four hidden units in the network. White squares within each of these images represent excitatory weights, black squares represent inhibitory weights, and grey squares represent weights with small magnitude. Notice that the connections from the pixels in the upper corners of the image to each of the hidden units have relatively large magnitude, indicating the network is depending heavily on the values of those pixels to determine the correct direction to steer. The reason the network came to rely heavily on those pixels is that during training, no guardrails or passing cars appeared in the periphery during training. In the case of the left periphery, it contained only grass over the entire stretch of road the network was trained on. As a result, the size of the dark triangular patch of grass in the upper left corner was a good indication of how sharply the vehicle should turn. The larger the dark patch, the more towards the left the vehicle was, and therefore the more the vehicle should turn to the right. Because of the high correlation between the size of the dark patch and the correct steering direction, the network learned to use this feature to determine how to steer. Therefore, the connections from the upper left pixels were given large weights, while the center and bottom portions of the image were largely ignored. The same argument hold true for the pixels in the upper right corner: the network relies heavily on them because of their consistency during training.

The quantitative effect transitory image features like guardrails have on driving performance is shown in the leftmost two bars of the graph in Figure 4.5. These two bars represent the performance of a network trained using the technique described in the preceding chapter on a sequence of multi-lane test images like the ones

[1]The yellow line appears dark in the preprocessed image because of its relatively small blue components (see Chapter 2 for details).

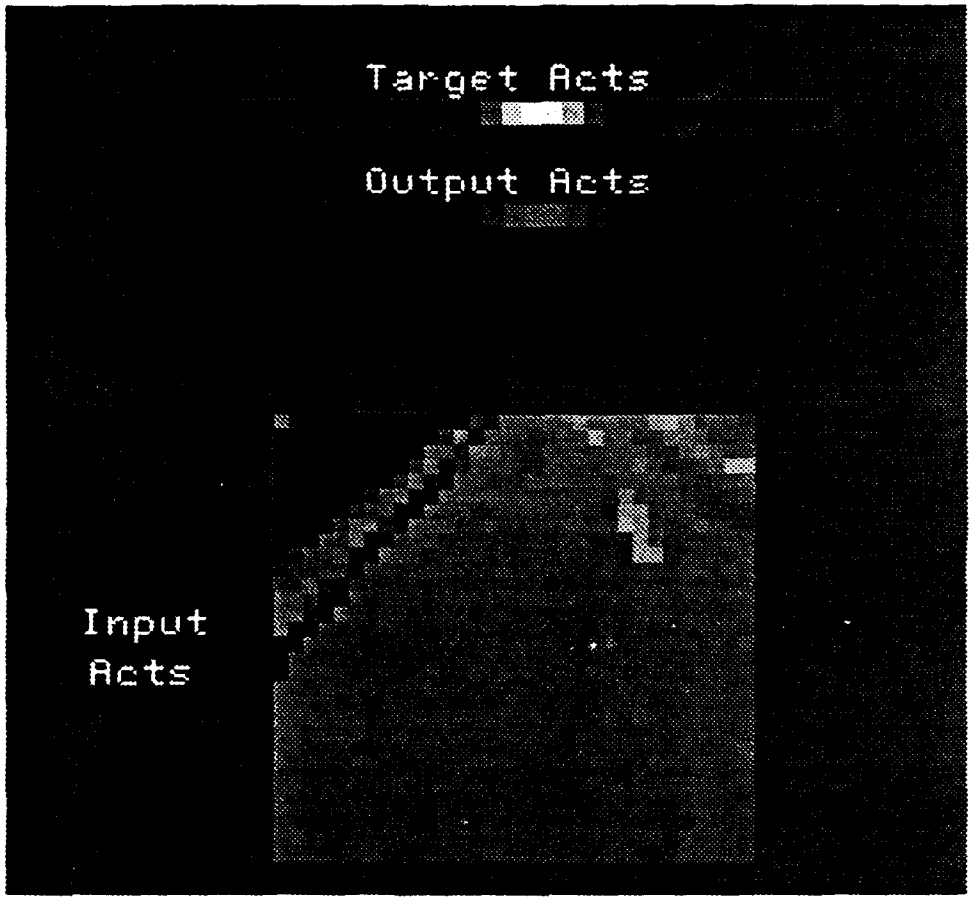


Figure 4.2: A typical two low resolution multi-lane highway image, and the response of a network trained without noise. The units with high activation in the network's output are in the same location as the high activation units in the target vector, indicating the network is steering in the correct direction.

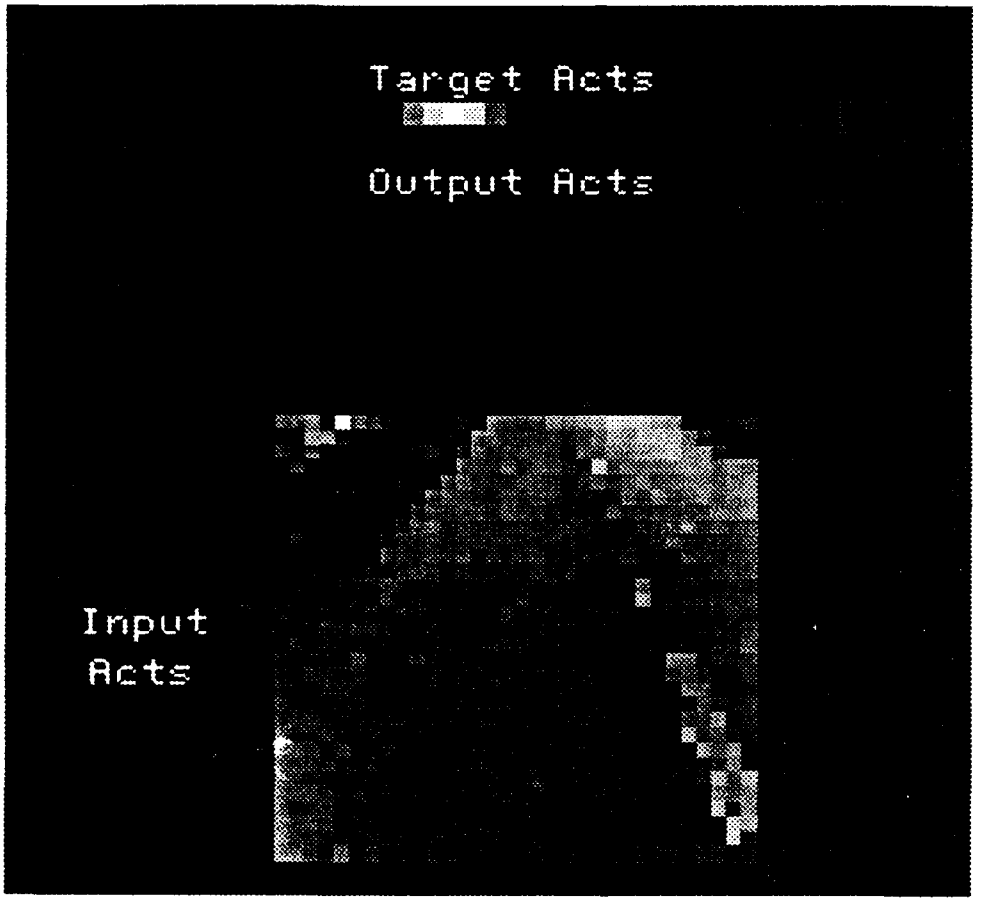


Figure 4.3: A low resolution multi-lane highway image in which a guardrail appears on the left side of the road (the bright triangular region in the upper left corner). Since the network did not see a situation like this during training, its steering response is far from correct.

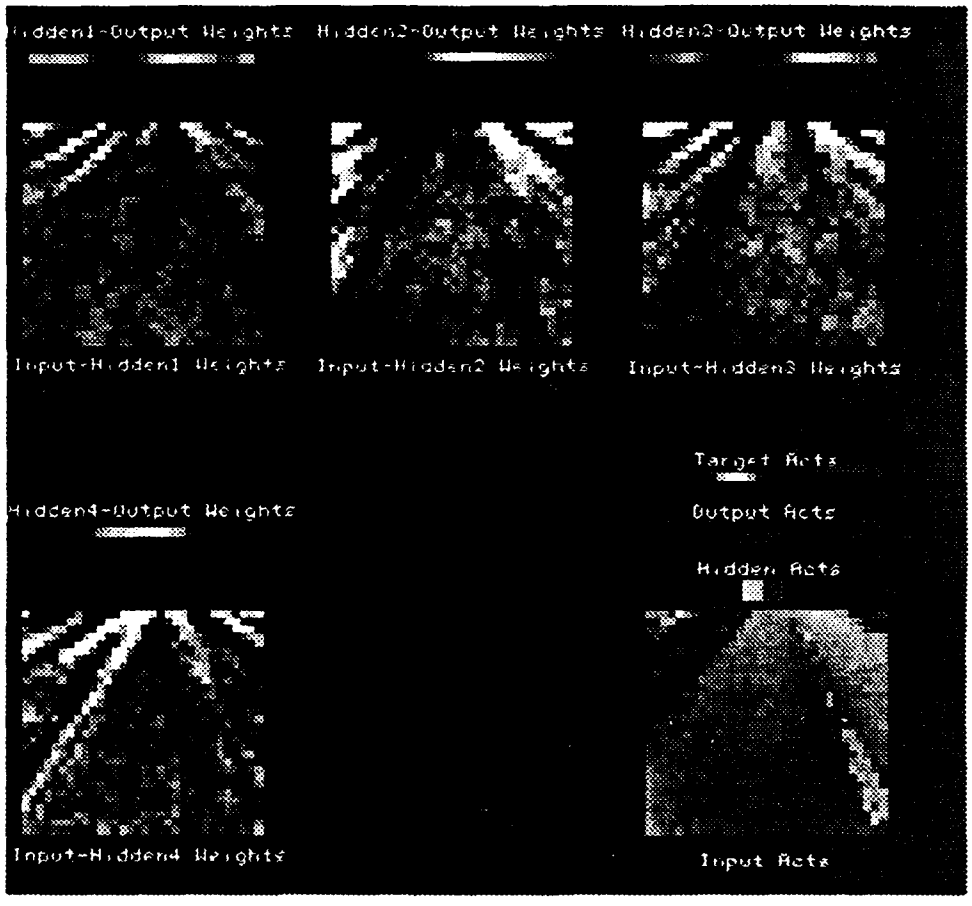


Figure 4.4: Weight diagram of a network trained without noise. The blocks labeled Input-Hidden1 Weights through Input-Hidden4 Weights represent the magnitude and sign of the weights from the input retina to each of the four hidden units in the network. The dark a square within each of these blocks the more inhibitory the corresponding weight. Conversely, the lighter a square, the more excitatory the corresponding weight. The upper corners of each of these input-to-hidden weight diagrams illustrate that the network has learned to rely heavily on features in the periphery to determine how to steer. As a result, when these features are corrupted, such as when a guardrail appears, the network's response is significantly disturbed.

shown in Figure 4.1. The leftmost bar shows the network's average steering error over the entire test set. Recall that steering error represents the curvature difference between the steering arc suggested by the network for an image and the arc the person was driving along when the image was taken. The bar next to it, labeled "Images w/ Guardrail" shows the average steering error of the same network on a 32 image subset from the test sequence which contained a feature not present in many of the training images, namely a guardrail on the left side of the road. The network's steering error more than doubles on images containing the novel feature, clearly illustrating the effect these features can have on performance. If allowed to steer autonomously over the stretch of road where the guardrail images were taken, the network would steer off the road and crash. Clearly something must be done to correct this behavior.

The problem caused by transitory image features is analogous to the problem addressed by training "on-the-fly", namely insufficient diversity in the training examples. When training on-the-fly with just the images coming directly from the camera, the system doesn't see any situations where the vehicle is off the center of the road. Therefore, it doesn't learn to recover from mistakes. By shifting and rotating the training images, the diversity of the training set is increased, resulting in a network better able to recover from steering mistakes.

In the case of transitory, spurious feature changes, the network does not encounter them frequently enough during the short training period to learn to ignore them. One useful aspect of the problem, exploited in the solution below, is that these transitory features do not radically alter the overall appearance of the image. While they may obscure or replace features which appear in a "normal" image, there remain enough features in the image to at least in theory make driving using the same network possible. In other words, because of redundancy in the image, if the network could learn to ignore spurious features such as guardrails, it should still be capable of driving accurately. This is an important distinction, since there are situations, such as one vs. two lane roads, where all the relevant image features are sufficiently different to warrant training separate networks.

4.2 Training with Gaussian Noise

A commonly employed technique for improving generalization from a limited amount of training data is to add uncorrelated gaussian noise to the training patterns [Seitzma & Dow, 1991]. The idea is that adding noise to the input prevents

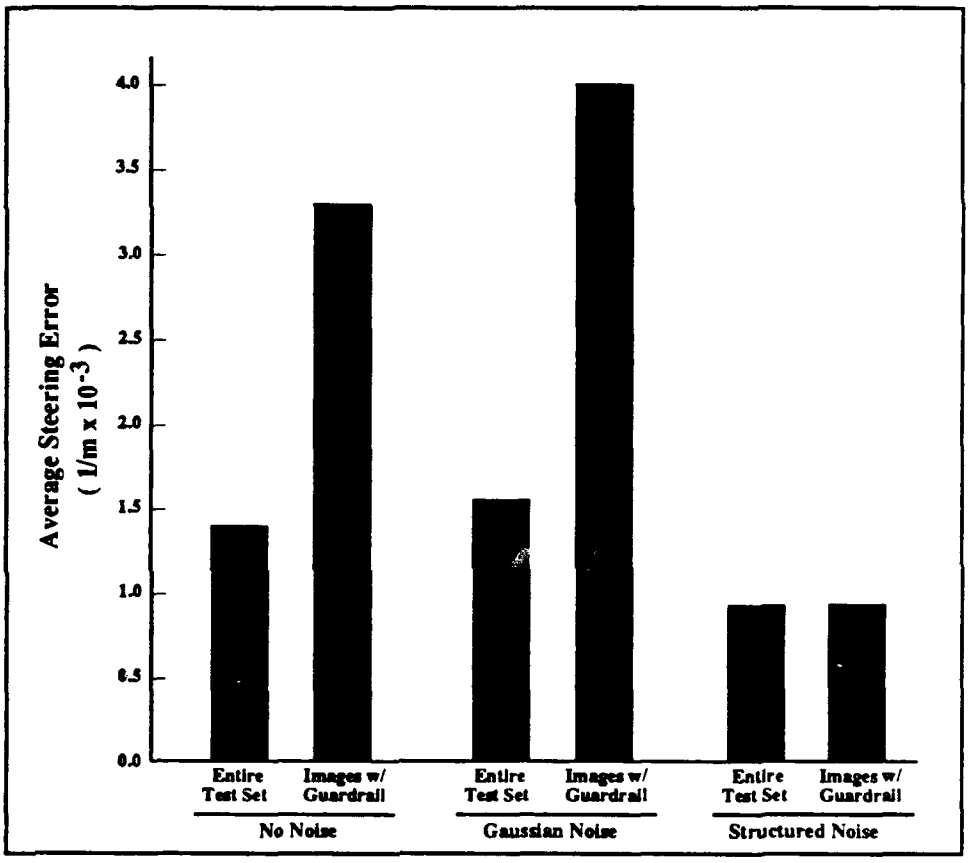


Figure 4.5: Graph illustrating the performance of networks trained using three different techniques on a test set of 180 images, and on a subset of 32 images containing a guardrail. The bars labeled "no noise" represent the average steering error of a network trained without noise. The bars labeled "gaussian noise" represent the average steering error of a network trained by adding gaussian noise to the input images. The bars labeled "structured noise" represent the average steering error of a network trained by adding structured noise to the input images.

the network from relying on idiosyncrasies in the training patterns to perform the task. Sietsma and Dow found a dramatic improvement in generalization when noise was added to the training patterns on a frequency classification problem. In their task, the input was a 64 unit vector whose input activation pattern formed a sine wave of a particular frequency. The task was to classify an input sine wave according to its frequency, regardless of its phase within the input field. They found that when gaussian noise was added to the training patterns, the resulting network made dramatically fewer classification errors on novel, noise-free input patterns (0.5% vs. 17%).

I performed the same experiment by training networks on the identical sequence of images as the noise-free network, but with the addition of various amounts of gaussian noise to the road images. The noise added to the training patterns for each network had a mean of 0 and a standard deviation ranging from 0.4 to 1.2. Figure 4.6 shows one corrupted road image used for training.

The results obtained by testing the noise trained networks on novel images were initially surprising. Regardless of the amount of noise, the networks trained with noisy input performed uniformly worse than the network trained without noise. This poorer performance is illustrated in the second pair of bars in Figure 4.5 labeled "Gaussian Noise". They represent the *best* performance of any of the networks trained with gaussian noise. Both on the entire training set, and particularly on the guardrail subset of images, the networks trained with noise steered less accurately than the network trained without noise.

The reason for this unexpected drop in performance is evident in Figure 4.6. Notice that the finer image features such as the lane markers, which were visible in the noise-free image of Figure 4.2, have been obscured by the noise. In fact, the only remaining distinct feature is the large patch of grass in the upper left corner. Since the size and location of the grass patch were the only image characteristics that the networks could accurately estimate, it was these that the noise-trained networks learned to key on to determine the steering direction. This degraded performance on the test set because despite its large size, the grass patch is actually a less reliable feature than the lane markers, since it is frequently obscured by guardrails and jersey barriers. Adding uncorrelated gaussian noise to the input has a tendency to obscure fine features like the lane markers, while leaving larger, more redundant features intact. When smaller features are more reliable indicators of the correct response than larger features, the result of adding gaussian noise during training will be poorer generalization.

Gaussian noise is a poor model of the important "noise" that occurs in the input

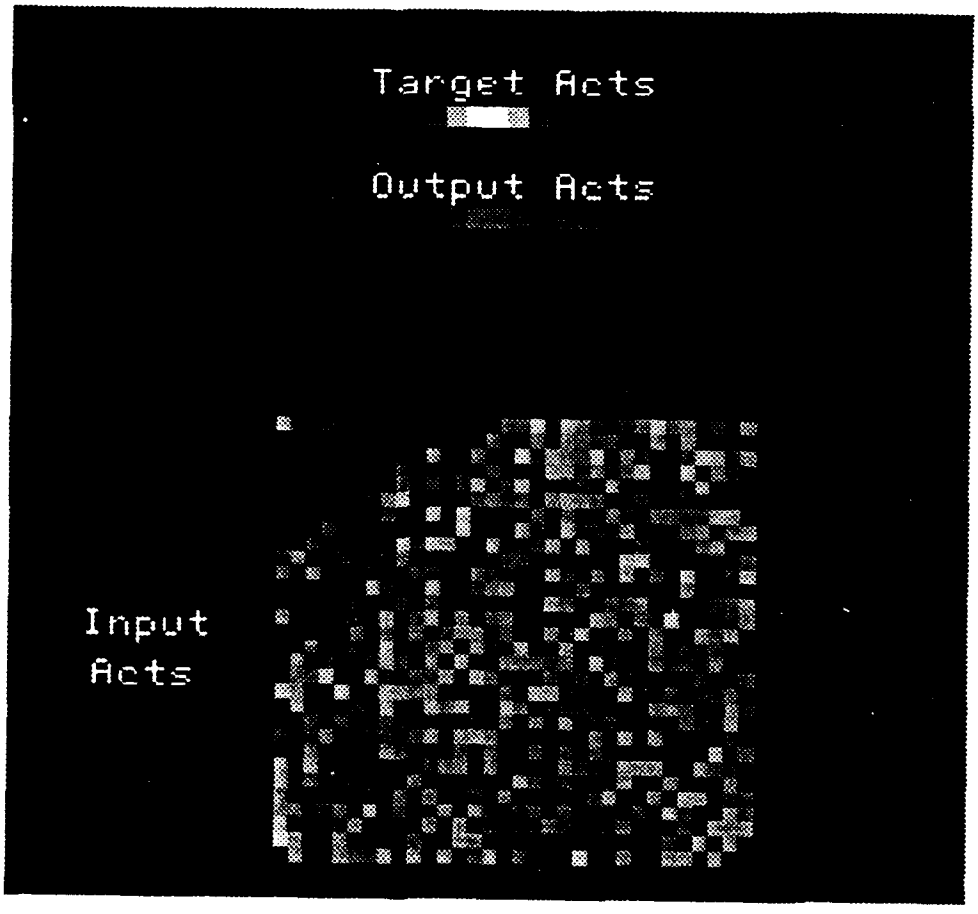


Figure 4.6: A multi-lane highway image corrupted with gaussian noise. Notice that the finer image features, such as the lane markers, are obscured by the noise, while the large patch of grass in the upper left corner remains visible.

while driving. The noise that matters results from the appearance or disappearance of coherent 2-D features such as guardrails and other cars which do not consistently correlate with the correct response. Modeling these irrelevant features and adding them to the input while training can improve generalization dramatically, as illustrated by the last two bars in Figure 4.5. Networks trained by adding structured noise to the input, as described in the next section, generalized better on the test set as a whole than networks trained without noise or with gaussian noise. Even more significant was the performance improvement of the network trained with structured noise on images with spurious features in them, as illustrated by the low error of the network on the guardrail images. In fact, the network's steering performance on the novel guardrail images was not significantly different than its performance on the test set as a whole, demonstrating that it has learned to ignore spurious image features (See Figure 4.5.

4.3 Characteristics of Structured Noise

A number of straightforward characteristics of structured image noise can be employed to improve network generalization. The most obvious features is the high degree of spatial correlation. Important noise does not appear as corruption of random image pixels. Instead, the physical object (like a car or guardrail) causing the noise makes a 2-D projection into the input image. As a result, the important image noise takes the form of spatially coherent two-dimensional regions of nearly uniform intensity. While objects with multiple or variable intensities do occur, their rarity makes a uniform intensity model of structured noise a reasonable approximation.

A more subtle characteristic of structured image noise results from the fact that objects can change in three ways. Objects can suddenly appear in the image, obscuring part or all of a previously visible feature. An example of this effect is when a car passes and obscures the lane markers. Objects can also disappear from the image, making previously hidden features visible. This effect might occur when the guardrail disappears, revealing the grass on the other side. Finally, a feature can change color or brightness, as when the centerline changes from white to yellow. Shape is not considered a changing feature characteristic, since objects visible when driving are assumed to be rigid.

A useful domain-specific characteristic of structured image noise is that when driving, irrelevant features are more likely to occur in the periphery of the image

than in the center. Even if during training the appearance of the terrain off to the side of the road remains constant, it is helpful for the network to learn that the appearance of the periphery can change dramatically. This way the network can learn not to rely heavily on the position of the guardrail or the size of the grass area in the off-road region, since they may be obscured or disappear at any time.

The size of significant image features can be constrained by estimating the size and distance to features which may appear or disappear in the scene. The retina size of significant features ranges from a few tens of pixels for the lane markers or guardrail to several hundred pixels for large objects such as passing trucks. The *shape* of significant image features is also strongly constrained in the domain of autonomous driving. In particular, image features are frequently oriented with their primary axis pointed towards the vanishing point of the image. The technique for incorporating these characteristics of image features and the noise that corrupts them into the training process is described in the next section.

4.4 Training with Structured Noise

Structured noise characteristics are used during training to determine the appearance of the noise to be added to the input patterns. Instead of adding gaussian noise to each pixel, the following technique is employed to add or remove coherent two-dimensional features to the training patterns.

First, for each pattern on each epoch of the back-propagation, a decision is made whether to add noise to that pattern or not. Instead of adding structured noise on every presentation of a pattern, I've found that occasionally presenting each training pattern without noise helps to maintain the network's ability to handle clean images. Empirically, adding noise on 3 out of every 4 presentations of a pattern seems to be a good compromise between teaching the network to be insensitive to noise and teaching it to process clean images correctly.

Once the decision is made to add noise to a pattern, the next question is where to put it. I have found that when trained on images with a single noise feature, a network is able to generalize to images with multiple noise features. Therefore, at most one noise feature is added per image during training. The location for this single noise feature is selected randomly with a bias towards the periphery of the scene, corresponding to the upper corners of the image. That is, the likelihood that a pixel will be chosen as the starting point for the noise feature is proportional to its proximity to the closer of the two upper corners of the image. This periphery

bias roughly models the tendency of noise features to appear away from the path directly ahead of the vehicle in real world driving situations.

After determining the starting location for the noise feature, a decision is made whether a new feature should be added at that position or an existing feature should be removed from that position. This choice is made randomly, but with a bias towards deleting a feature if one appears to exist at that location, and towards adding a feature if that location appears to be "feature free". The judgement concerning whether a feature exists at a location is made based on the size of the uniform region that location is part of. First, a region is grown around the chosen location to encompass all the contiguous pixels whose intensity is within a fixed threshold of the chosen pixel's value. If the size of this region is within the size range of interesting features as characterized above, then the location is considered to be part of a feature and that feature is removed. If the size of the region falls outside the size range of interesting features, it is considered part of the background, and a new feature is added at that location.

Removing a feature is easy. The pixels defining the feature have already been determined in the region growing step described above. Deleting the feature involves changing the values of all the pixels within the region to a new randomly chosen intensity. This alteration of a region's intensity models the corresponding object changing color. By selecting the new intensity to be the same as the intensity of the area surrounding the feature, the disappearance of the feature can also be simulated.

The result of using this technique to alter an existing feature is illustrated in Figure 4.7. The image is identical to the one in Figure 4.2, except that the patch of grass in the upper left corner has changed intensity from very dark to very light.

Adding a new feature to model the sudden appearance of objects such as a guardrail or automobile is more difficult, since unlike in the feature elimination/alteration process described above, the shape of the feature is not known. The shape of spurious 2-D image features can in theory be arbitrary. There is the weak constraint mentioned earlier that significant features tend to have their major axis pointing towards the vanishing point. However, this orientation specificity is difficult to implement directly in the noise generation model since it requires knowledge of the sensor geometry.

This sensor geometry information is available, since it is used in the image transformation scheme described in the last chapter, but there is a simpler and more general technique for modeling feature shape. The technique involves using the shape of the feature detectors the network develops in its internal representation

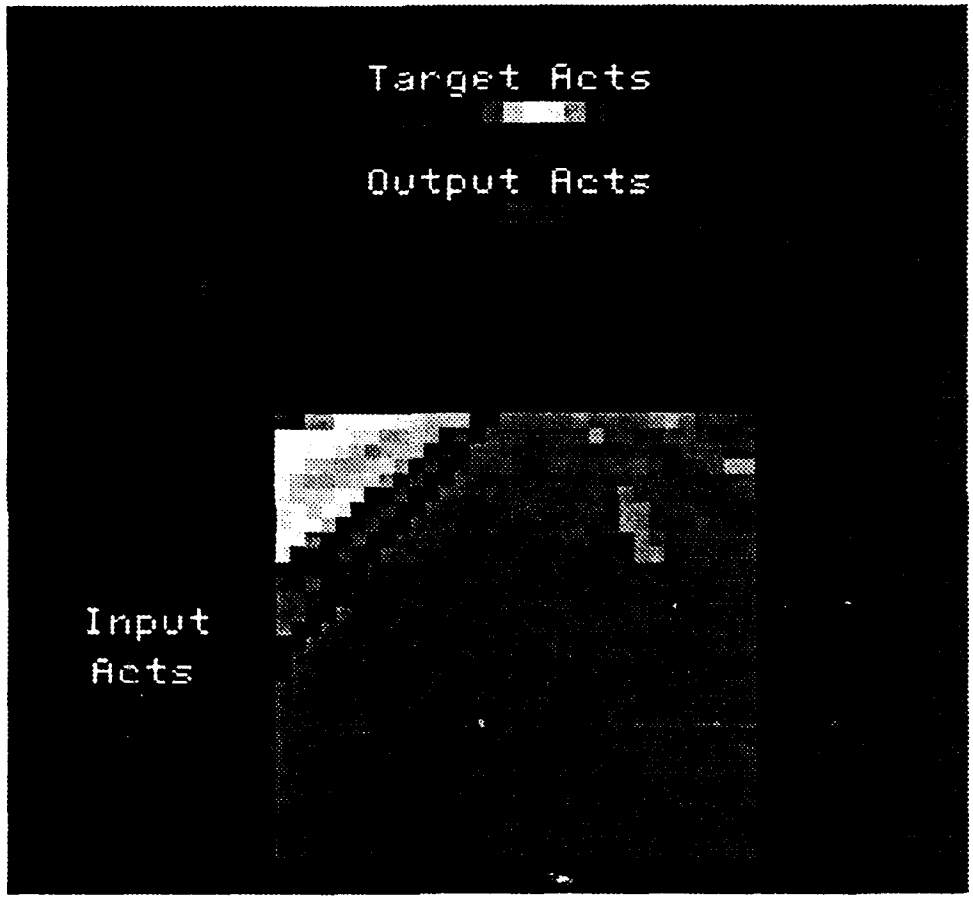


Figure 4.7: An image in which an existing feature has been altered by changing its intensity. The image is identical to the one in shown earlier, except that the patch of grass in the upper left corner has changed intensity from very dark to very light.

to bias the shape of the noise features added as distractors. As is evident in the weights from the input retina to the hidden units in Figure 4.4, the network develops a strong model of the shape of important image features. The network's knowledge that features tend to be oriented towards the vanishing point is demonstrated by the manner in which the features in the receptive fields of the hidden units converge towards the top.

To bias the new feature's shape using the shapes of the features in the network's internal representation, a random hidden unit is first selected. The values of the weights from the input retina to this hidden unit are then used as the image in which to grow the new feature. In other words, a pixel in the vicinity of the feature's start pixel is chosen to be an element of the feature if the weight of the connection from it to the chosen hidden unit is sufficiently similar in value to the weight of the connection from the feature's start pixel to the chosen hidden unit. By biasing the shape of noise features by the shape of important features according to the internal representation, this technique ensures that only noise features with a reasonable shape are added to the input. In addition, a noise feature whose shape corresponds to the contours formed by a coherent group of large magnitude connections from the input retina guarantees that noise feature will have maximum impact for its size. This is because if the noise feature stretches "across the grain" of the receptive field, the influence of the pixels connecting to the chosen hidden unit with large negative weights would cancel out the influence of the pixels with large positive connections. This would result in the noise feature having little net influence on the hidden unit's final activation level.

To mimic situations in which image noise does not precisely align with significant feature detectors, a spatial coherence constraint is added to the feature growing algorithm. The following equation balances the tendency of the feature growing algorithm to follow hidden unit weight contours with the tendency to create a spatially compact feature. A pixel i will be included in a newly created features if:

$$\alpha|w_{ih} - w_{sh}| + (1 - \alpha)|i - s| > T$$

In this equation, w_{ih} is the weight value of the connection originating at pixel i in the input image and terminating at the chosen hidden unit h. The variable w_{sh} corresponds to the weight of the connection originating at s, the pixel chosen as the start position of the feature, to the chosen hidden unit h. The quantity $\| i - s \|$ represents the Euclidean distance between pixels i and s in the input retina. The

constant α is used to weight the tendency to follow hidden unit weight contours with the tendency to keep the feature spatially compact. With an α of 0, the new feature will form a circular region centered on pixel s. With an α of 1, the new feature will grow to include all the pixels in the image having connections to hidden unit h with a weight close in magnitude to the connection from pixel s to unit h. Randomly choosing an α from the range 0.1 to 0.3 for each image creates realistic looking new noise features. The threshold T is used to limit the growth of the new feature. If the weight from pixel i to hidden unit h differs significantly from the weight from the start pixel s to unit h, or if pixel i is a great distance from pixel s, then the threshold T will be exceeded by the above sum and pixel i will not be included in the new feature.

Once the position and shape of the noise feature is determined, it is made to contrast with the surrounding area by filling it with a randomly chosen uniform brightness. A real video image in which a noise feature (the dark region in the upper right) has been added is shown in Figure 4.8. Notice how the feature appears close to the upper corner of the image due to the periphery bias built into the noise feature generation algorithm. Figure 4.8 also illustrates how growing noise features along important contours in the hidden unit representation results in features with appropriate orientations for the domain. In this example, this bias results in a feature oriented diagonally towards the vanishing point. The feature is spatially compact due to the bias in the noise generation algorithm towards choosing pixels in the vicinity of the feature's start pixel. This combination of biases based on known, consistent image feature characteristics results in the generation of noise features with realistic appearance. In fact, the feature in Figure 4.8 appears very much like a car passing by on the right side of the vehicle, as in the right image of Figure 4.1.

As illustrated in the bar graph of Figure 4.5, a network trained by adding and removing features steers more accurately than a network trained without noise, particularly on images containing spurious features. The reason for this is evident in the weight diagram of Figure 4.9. The network shown in this diagram was trained on the same sequence of images as the network shown in Figure 4.4, except that on each iteration of back-propagation, 75% of the patterns were randomly selected to have a noise feature added to their input. Varying the noise at every epoch prevents the network from learning characteristics of a single noise feature. The remaining 25% of the patterns were presented without noise to ensure the network would also handle noise free situations.

It is clear by looking in the upper corners of the input-to-hidden weight arrays

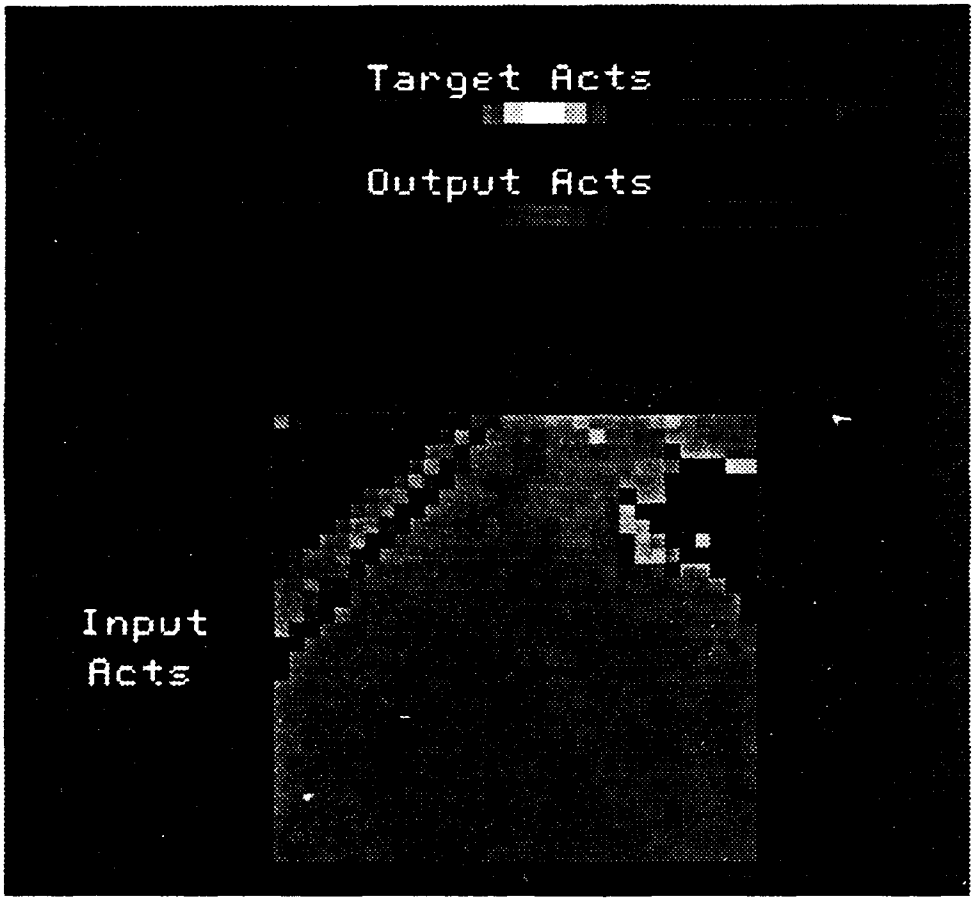


Figure 4.8: An image in which a new feature has been added by growing a region in the weight array from a single hidden unit, subject to a spatial compactness constraint.

that adding structured noise to the image has had the desired effect. Namely, the network has learned to rely less on features in the periphery than the network depicted in Figure 4.4. In fact, the network has developed detectors primarily for determining the position of the line marking the left lane boundary, since this is the feature that appears most reliably in this type of road image. Because of its frequent occlusion and absence from the image, the dashed line marking the right boundary of the lane is given little importance in the internal representation. The resulting performance improvement is illustrated by the input pattern and corresponding network response in the lower right corner of Figure 4.9. The input pattern is the same guardrail image that confused the network trained without noise in Figure 4.3. This network handles the guardrail image perfectly, since it has learned not to be disturbed by features in the periphery.

4.5 Improvement from Structured Noise Training

The bar graph in Figure 4.5 illustrates the significant increase in steering accuracy which results when structured noise is added in the training process. This steering accuracy improvement translates directly into dramatic gains in driving performance. Quantitatively, I have found that a network trained without the addition of structured noise on a two mile stretch of a four lane divided highway was capable of driving autonomously for only about four miles before straying from the road because of a spurious feature. Driving successfully for even this relatively short distance was somewhat fortuitous, in that it was achieved on a stretch of road free from guardrails and other potentially confusing permanent features, and at a time of day when there was very little traffic to confuse the network. In fact, when other vehicles did appear during these tests, the network trained without noise frequently swerved towards or away from them depending on their brightness relative to the background. The swerves were relatively small, so the safety driver allowed the run to continue without interference. The situation which ended the run after four miles was the one depicted in the center image of Figure 4.5. The vehicle encountered a bridge with jersey barriers along the edge of the road. This spurious feature caused the vehicle to swerve dramatically, forcing the safety driver to intervene.

The network trained with structured noise drives significantly better. Its best run was 21.2 miles without human intervention, on the same highway that caused the network trained without noise to fail after 4.0 miles. It made it over the bridge that caused the previous network to fail, and through a number of other situations

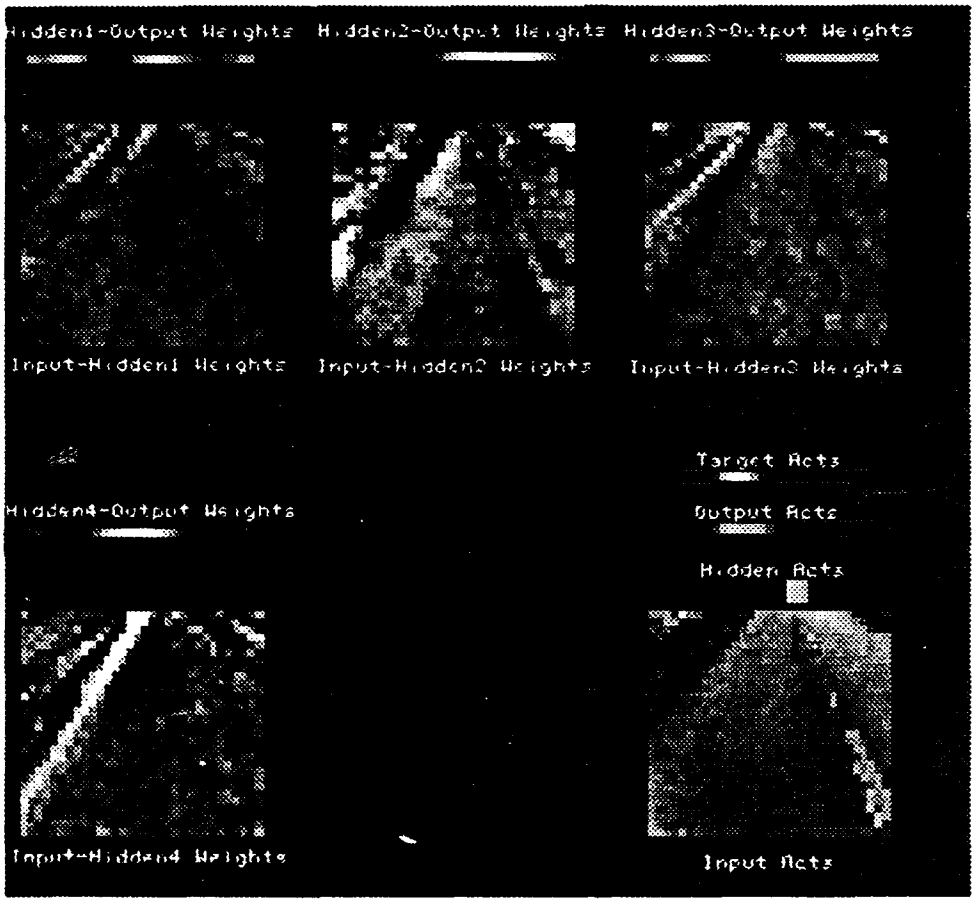


Figure 4.9: Weight diagram of a network trained with structured noise. As is evident by the low magnitude of the weights from the upper corners of the image to the hidden units, adding structured noise has taught the network not to rely on features in the periphery, since they are not a reliable indicator of the correct steering direction.

that would have caused trouble for the network trained without noise. The reason the run came to an end after 21.2 miles was that the width of the road changed significantly, causing the network trained with noise to become confused. For more details on the performance of this network, which actually set a new world distance record for autonomous navigation with its 21.2 mile run, see the next chapter.

4.6 Discussion

The appearance or disappearance of irrelevant features can disrupt a network's driving when the network's training did not demonstrate their irrelevance. Adding structured noise to the training patterns using a model describing the characteristics of irrelevant features significantly improves driving performance. A simpler gaussian model of image noise has been shown to be ineffective at compensating for this problem because it does not mimic the important characteristics of real world, structured noise.

But are there other possible solutions which do not require such complex modeling? In theory, training the network over a longer stretch of road than the two miles currently employed should result in a more representative training set and hence a more robust network. However there are a number of shortcomings to this approach. One long term goal of this work is the development of a super cruise control system capable of controlling both the vehicle's speed and steering. If the training period for this super cruise control is too long, it will be impractical.

But even given unlimited training time, the scarcity of irrelevant features would make it difficult to train a network to ignore them. For instance, over many miles of highway driving, the size of the patch of grass on the left side of the image is a good indicator of the correct steering direction. Only in rare situations, like going over a bridge, will relying on this feature get the network in trouble. Even if the training period were extended to ensure encountering this type of situation, its low frequency would make it beneficial for the network to ignore these few patterns. This is because the network could lower the total error over all training patterns by employing the patch of grass to improve the steering performance on the vast majority of patterns, while suffering substantial error only on the few patterns in which the patch of grass is missing. In other words, the high correlation between an irrelevant feature and the correct output would result in the network employing the feature despite being exposed to a few situations where this degrades performance.

It might be possible to achieve a similar result by intelligently selecting patterns during training. The network could be trained solely with images on which it disagrees with the person's steering response. A similar effect could be achieve with a back-propagation error term that penalizes large steering errors more than small ones. These techniques, would increase the impact of rare circumstances in which irrelevant features cause the network to make a large mistake.

There are three problems with these alternative methods. First, the human driver does not always drive attentively and therefore occasionally steers incorrectly. Techniques which emphasis discrepancies between the network and the human trainer will encourage the network to imitate the person's mistakes, clearly an undesirable effect. In addition, these technique require the vehicle to encounter a situation at least once in order for the network to learn about it. For instance, if a bridge never appeared during training, the network would still mistakenly learn to rely on grass to the left of the road. In contrast, by employing a model of structured noise during training, the network can learn about this situation which the vehicle never actually encounters. Related to the first shortcoming, a significantly longer training time would be required to ensure that the network has encountered all the possible types of spurious features. From a practical standpoint, it is much more straightforward to build a model of the noise and add it directly.

But is there a simpler way of building the domain knowledge currently embodied in the structured noise generation process into the training procedure? There very well may be. One possibility would be to constrain the network architecture by limiting its connectivity in order to force the network to ignore irrelevant features. The problem with this approach is that it is difficult to determine the exact appearance and location of irrelevant features before the training process begins. In autonomous driving, features in the periphery are known to be less reliable than more central features, but the actual impact of this general rule on the images depends on the camera geometry and the particular features of the domain. The technique for adding structured noise overcomes this problem by applying the general rule of peripheral unreliability on an image by image basis. Instead of predetermining which pixels in the image will be ignored by limiting the network's connectivity pattern, the structure noise generation technique disrupts entire peripheral features in individual images, allowing the network to learn on its own the position and appearance of irrelevant features.

The current model of structured image noise has obvious shortcomings. Noise features do not always occur in the periphery. When driving in traffic, cars directly in front of the vehicle will obscure central image features. But even this simple

model is sufficient to demonstrate an important point: dramatic improvements in network generalization can be achieved by actively employing domain-specific knowledge during the training process. Both the technique described in the previous chapter for training on-the-fly and this technique for adding structured noise demonstrate that domain knowledge can be effectively exploited by augmenting the training set with patterns representing relatively simple transformations of situations actually encountered.

Chapter 5

Driving Results and Performance

The preceding three chapters focused on techniques that allow artificial neural networks to perform vision-based guidance of a mobile robot. Actual results employing these techniques have been included only for drawing comparisons between competing methods. In the first part of this chapter the flexibility of the techniques presented so far is demonstrated by illustrating the range of situations in which networks have been trained to drive. Accompanying the description of each situation is an analysis of the characteristics that made the domain interesting or difficult, and an outline of any situation-specific steps required to facilitate driving in that domain.

The remainder of the chapter is devoted to quantitatively characterizing the performance of the neural network driving system. Performance of any autonomous driving system is difficult to quantify because of the dynamic nature of the task. Despite this difficulty, I will describe one measurement technique I have developed, and use the technique to compare a network's performance with that of a human driver.

5.1 Situations Encountered

I will first describe situations that ALVINN has successfully handled using video camera input. I will then present situations in which the use of alternative sensors has allowed ALVINN to drive proficiently.

Figure 5.1: Video images taken on three of the road types ALVINN modules have been trained to handle. They are, from left to right, a single-lane dirt access road, a single-lane paved bicycle path, and a lined two-lane highway.

5.1.1 Single Lane Paved Road Driving

The first scenario to which I applied neural network based autonomous navigation, and the domain which has served as the proving ground for many of the techniques described in this dissertation, is driving along a 400m bicycle path through a wooded area adjacent to the CMU campus (see the middle image of Figure 5.1). Driving on this road is difficult for many reasons. First, the road contains five sharp curves, making rapid steering corrections extremely important. The need for precise and timely steering corrections is demanded by the narrowness of the road relative to the vehicle. The road averages 2.9 meters wide, which is only slightly more than the 2.55 meter width of the two test vehicles. In addition, the road is not actively maintained. As a result, the road's width and appearance varies substantially due to the effects of missing or broken sections of pavement. Depending on the season, the existing pavement is also frequently obscured by fallen leaves or snow. Another difficulty results from the fact that the road traverses both a grassy field and a wooded area with only dirt surrounding the road. This change of terrain results in a dramatic change in the appearance of the off-road region in color video images. In addition, the trees bordering the road in the wooded area often cast harsh shadows, creating spurious features in the image. Finally, the road contains three intersections which can be confusing for an autonomous driving system.

Despite the inherent difficulties with this domain, a network can learn to steer on this road in about 3 minutes by watching a person drive along its. After training,

the network can generalize to drive the road in the opposite direction at speeds of up to 10 mph. The ability to drive in the other direction illustrates that the network does not learn idiosyncrasies about the appearance or turn sequence of this particular stretch of road, but instead learns general image features useful for driving on this *type* of road.

In addition to generalizing to novel stretches of this type of road, a network trained on this road can also drive in weather and lighting conditions it never encountered during training. This ability has been demonstrated by training a network one day, and then testing it the next after conditions have changed. A single network trained under either sunny, cloudy or rainy conditions can also accurately drive in the other situations.

The one climatic condition that has proven difficult to generalize about from real training data is snowy conditions. A network trained when there is snow covering the non-road areas of the image has no trouble driving in those conditions. However, a network trained under snowless conditions is unable to drive in snowy situations, and a network trained under snowy conditions is not able to drive in snowless situations. This failure to generalize results from the drastic appearance change associated with snow. Under conditions without snow, the road appears brighter than the non-road in the preprocessed image, since the road is bluer than the non-road (recall from Chapter 2 that the blueness of a pixel determines its brightness in the preprocessed image). The reverse is true under snowy conditions. That is, the road appears darker than the non-road in the preprocessed image.

This flip in polarity of the relative intensity of the road and non-road makes generalizing between the two situations difficult. I have successfully trained a single network to drive in both snowy and snowless conditions. However, this was achieved by using a batch training method, in which the network was trained on images collected from both situations instead of the normal "on-the-fly" method of training.

A network trained on live images of roads without snow doesn't generalize to snowy images since the training set employed during a live training run doesn't contain any patterns resembling the road under snowy conditions. However using a simple model describing how the image can flip polarity, the training set could be easily augmented to include these situations using a technique similar to the one described in the last chapter. In other words, instead of randomly adding feature noise to the training images, they could be occasionally inverted in order to teach the network that the road can be either darker or lighter than the non-road.

In fact, before I developed the technique for training on real images, the

artificial patterns I generated for training included both images in which the road was lighter than the non-road, and images in which it was darker. Again I found that by training with both types of images, a single network could learn to drive in both situations. However due to the added complexity associated with learning road polarity invariance, training took significantly longer. This extra training time, coupled with the fact that it is very rare that the road changes polarity with the non-road, prompted me to discontinue training networks to be invariant to this type of image change.

The more efficient solution I've developed to deal with widely varying conditions involves training separate "expert" networks for each, and then choosing the most appropriate network using connectionist arbitration techniques described in Chapters 8 and 9. By requiring a single network to only handle a relatively limited set of circumstances, training time is shortened and overall performance is improved.

5.1.2 Single Lane Dirt Road Driving

Another road type ALVINN has been trained to handle is unpaved single-lane roads. This road type is interesting for two reasons. First, on some of the unpaved roads ALVINN has been trained on, the contrast between the road and the non-road is much lower, and the amount of image noise is much higher than in the paved single-lane road domain. Despite these added difficulties, ALVINN can drive quite proficiently on these roads.

An interesting variant of unpaved road that an ALVINN network has be trained to drive on is defined only by parallel wheel ruts running through a uniform environment. On the road tested there was grass surrounding the road and also grass down the middle of the road between the two ruts. This configuration can confuse road following systems which rely on classifying pixels as part of the road or part of the non-road and then fitting a model of the road's shape to these classified pixels [Crisman & Thorpe, 1990]. The problem stems from the fact that the usual model of an unstructured road, as a trapezoidal image region differing in color from its surroundings, is violated in this situation since grass pixels occur in both the road and the off-road regions (for more details see Chapter 11).

In contrast, ALVINN does not have a strong predefined model of what important image features should look like. Instead, ALVINN is able to learn that it is the position and orientation of the two ruts in the image that determine the correct steering direction. This ability to learn a new road model allows ALVINN to drive

as proficiently on this type of road as on roads with a more typical appearance.

5.1.3 Two-Lane Neighborhood Street Driving

A domain that further illustrates the flexibility of the neural network approach to autonomous navigation is driving on two-lane, unlined suburban neighborhood streets. The significant distinctions between this road type and the previous ones are the added road width, and the frequent occurrence of driveways intersecting the road at right angles.

As in the previous domain, accurate driving is difficult to achieve on this type of road using techniques based on pixel classification and road model fitting for three reasons. Like the grass down the middle of dirt roads, the driveways in this domain make it difficult to accurately represent the road with a simple trapezoid model. In addition, the greater road width means that frequently only one of the road's edges will appear in the image. When one edge is missing, the model fitting process employed by classical road followers becomes more difficult. Finally, even after finding the position of the road, traditional model-based autonomous navigation systems would have to be programmed by hand to stay to the right on this type of two-lane road, instead of driving down the middle.

ALVINN learns to ignore the spurious image features caused by the driveways, and to keep the vehicle driving down the right side of the road simply by watching a person drive in this domain for about four minutes. ALVINN has successfully driven in this domain for 1/2 mile at speeds of up to 13 mph. By combining ALVINN's road following ability with the symbolic reasoning of an annotated map system, the system was able to navigate through three intersections over the 1/2 mile course, coming to rest at a preplanned destination. For more details about this run, and the hybrid connectionist/symbolic approach which made it possible, see Chapter 7.

5.1.4 Railroad Track Following

Another domain that demonstrates how ALVINN can learn to maintain the vehicle at an arbitrary position relative to features in the world is driving parallel to railroad tracks. In this domain, there was a very poorly defined road running next to a set of railroad tracks. Since, there were no consistent road features for the network to key off, driving using a camera pointed straight ahead was impossible. But the solution was easy. I simply turned the camera slightly to get the rails of the

railroad tracks in its field of view and trained the network by driving parallel to the tracks. The network learned in about 4 minutes to steer the vehicle in order to keeps the rails at a particular position and orientation in the image. Again, because of the network's flexibility, this domain was no more difficult to master than the previous ones.

5.1.5 Driving in Reverse

A driving situation which, like railroad track following, required a non-standard camera positioning was driving in reverse. By mounting a camera on the rear of the vehicle facing backwards, and training a network while a person backed up, ALVINN learned to drive in reverse on the single-lane paved road described above.

The interesting aspect of this domain is that ALVINN was actually better at driving backwards than its human trainer. There were two reasons for this improvement. First, because of people's bias towards driving forward, the human trainer could not get accustomed to driving in reverse using the rear facing camera. His difficulty resulted from the fact that the rear facing camera produced a mirror image of the scene behind the vehicle when displayed on a monitor next to the steering wheel. If the road appeared on the right side of the image, his natural reaction was to turn right, instead of the correct response, left. So the human trainer continued to use the rear view mirrors while training the network.

Since an untrained ALVINN network has no bias about what to expect from the scene, it had no difficulty learning the inverse relationship between feature position and steering response required for driving in reverse using the rear mounted camera. As a result, ALVINN was able to drive at up to 10 mph in reverse just as proficiently as it could drive forward on the single-lane paved test road. The human driver, with the limited field of view provided by the rear view mirrors, was able to keep the vehicle centered on the narrow road reliably only up to a speed of 7 mph.

5.1.6 Multi-lane Highway Driving

The domain in which ALVINN's most impressive results have been achieved is multi-lane highway driving. Some of the difficulties in this domain, in particular the problems associated with spurious image features, were presented in the last chapter. Other factors that make highway driving a challenge are the high

speed required to keep up with the prevailing traffic, and the very subtle steering corrections required to keep the vehicle on the road without losing control.

Two slight modifications were required to enable ALVINN to drive in this domain. The first change was to increase the lookahead distance for steering to 45 meters. This change was made in accordance with the rule of thumb from Chapter 3 describing the relationship between driving speed and lookahead distance. Specifically, the lookahead was set to the distance the vehicle would travel in 2 second at the desired speed of 50 mph.

The second modification was to limit the magnitude of ALVINN's sharpest steering direction from the 20m turn radius used in the other domains to a relatively shallow 100m radius turn. Limiting the steering range serves two purposes. First, it provides greater resolution over the reduced steering range, permitting finer distinctions to be make. Second, it reduces the da-ger of suddenly swerving into another lane by preventing ALVINN from turning too sharply.

With these modifications, ALVINN was able to learn from two miles of training to drive autonomously down the left lane of a highway at a speed of up to 55 mph. The left lane was chosen to avoid the confusing effects of exit ramps. The speed was limited not by the network's performance, but by the physical limitations of the NAVLAB II, which won't drive more than 55 mph for any extended period, and by the Pennsylvania speed limit.

The network made two runs of slightly over 10 miles, and a world record run of 21.2 miles. This nearly doubled the previous world distance record for autonomous navigation, held the Dickmanns' group [Dickmanns & Zapp, 1987] driving along a 12 mile, newly paved and painted stretch of the German autobahn closed to other traffic at the time. ALVINN's 21.2 mile run was on a relatively old stretch of highway which was open to other traffic. During the record setting run, the average speed was 45 mph. Over the 21.2 miles, 14 cars passed the vehicle while it was being driven autonomously, without disturbing the network. The reason the safety driver finally had to intervene was that the width of the road changed significantly, confusing the network.

5.2 Driving with Alternative Sensors

The previous section demonstrated the flexibility of the connectionist approach to vision-based robot guidance by presenting the wide range of driving situations successfully handled by networks using video input. This section further illustrates

Figure 5.2: Images taken of a scene using the three sensor modalities the system employs as input. From left to right they are a video image, a laser rangefinder image and a laser reflectance image. Obstacles such as trees appear as discontinuities in laser range images. The road and the grass reflect different amounts of laser light, making them distinguishable in laser reflectance images.

the flexibility of the neural network approach by presenting ALVINN's ability to employ alternate sensors to drive in situations which would be impossible using only a video camera.

The primary additional sensor on both the Navlab and Navlab II is a scanning laser rangefinder. It rapidly scans a laser beam over a 30 by 80 degree field of view in front of the vehicle and measures both the amount and the phase of the light reflected back from the scene. The amount of returning light from each point in the scene can be translated into a two dimensional image representing the reflectivity of each point. The laser beams "time-of-flight" can be translated into an image representing the distance to points in the scene.

Images of the same scene created using the three sensor modalities are illustrated in the Figure 5.2. The scene contains a single lane paved road winding between two trees. In the video image on the left and the reflectance image on the right, the road is clearly distinguishable from the non-road region. In the video image, this distinction is created by the difference in brightness between the road and the non-road. In the laser sensor image, the road is distinguishable from the non-road because each reflects a different amount of light back to the sensor. In the range image (shown in the middle of Figure 5.2), the two trees on either side of the road appear darker than the points around them, because the trees are closer to the sensor; the points around them, which lie on the ground plane, are much farther away.

5.2.1 Night Driving Using Laser Reflectance Images

As can be seen in Figure 5.2, laser reflectance images resemble black and white video images. In fact, it was trivial to replace ALVINN's normal video input with laser reflectance input and train a network to drive on the single lane test road shown in Figure 5.2. All it required was resetting the parameters in the image transformation code describing the sensor's field of view to ensure that the shifted and rotated patterns created while training on-the-fly appeared realistic. Other than that, training of the network to drive using laser reflectance images proceeded exactly like training with video input. The person drove along a 300 meter stretch of the road while the network watched. After training, the network was able to take over and drive on its own.

The big advantage of using reflectance input over video input is that reflectance images appear the same regardless of the lighting conditions. In other words, since the laser sensor has its own light source, the reflectance images it creates are insensitive to ambient lighting. This allows ALVINN to be trained during daylight, when a person can see, and then tested in total darkness. A network so trained was able to drive at 6 mph at night with or without headlights along the paved single lane test road. Without headlights, the person drove less accurately than the network, since he had to rely solely on the limited amount of ambient light available. The reason ALVINN could not drive at the 10 mph speed achieved on the same road using video input was that the laser sensor only provides 2 images per second, while the video camera provides 30.

5.2.2 Training with a Laser Range Sensor

Employing laser range images to teach ALVINN about obstacles required more substantial changes than did reflectance images. The reason is that a different scheme must be devised to transform laser range images for training on-the-fly. For video and reflectance images, important images features are assumed to lie on the ground plane. Therefore, a simple, resampling of images pixels could be precomputed to transform the apparent position and orientation of the vehicle, as described in Chapter 3.

In range images, the important features (i.e., those corresponding to obstacles in the environment) violate the flat world assumption by extending above the ground plane. The consequence of this violation in the transformation process is illustrated in Figure 5.3. The image on the left shows the original range image

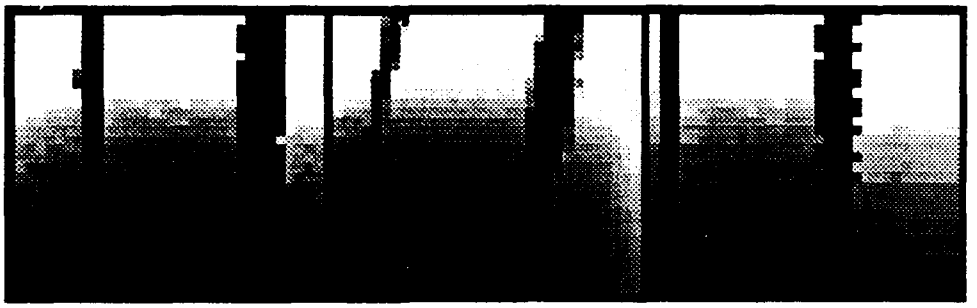


Figure 5.3: Three reduced resolution range images of a scene. The image on the left is the original range image. Notice the two dark vertical stripes caused by nearby trees. The middle image was created by transforming the image to make it appear 1 meter to the right of its original location, assuming the environment is planar. As a result of the flat world assumption, non-planar objects like the tree become unrealistically skewed in the image. The image on the right shows the same shift of the same image, but using the image's range information to accurately shift the pixels in the scene.

reduced to the low resolution employed by the network. The two dark vertical stripes result from nearby trees in the scene. The middle image shows the result of transforming the image to make it appear that the vehicle is 1 meter to the right of its location in the original image, assuming a planar world. Notice that pixels at the bottom of the image, corresponding to nearby points in the scene, are shifted more than pixels at the top, which are assumed to be far away. As a result of this differential shifting, the trees no longer appear vertical. Obviously, to create realistic new training exemplars from a non-planar scene, a different transformation scheme is required.

All the pixels corresponding to a single tree, regardless of their row in the image, should shift by the same amount since they are all approximately the same distance from the sensor. Fortunately, the distance to every point in the scene is easy to calculate, since it is simply a linear function of the corresponding pixel's intensity. In other words, the distance a pixel should be shifted in the image to make it appear that the vehicle has been displaced by a given amount can be calculated using only the pixel's value.

The equation that describes the distance a point will shift in a range image as a function of the vehicle's change in lateral position and heading, and the distance of the point from the sensor is given below:

$$S_i = \frac{C\,(\tan\Delta\theta + \frac{K}{I_i}(L\,\tan\Delta\theta + \Delta l))}{2\tan\phi}$$

where S_i is the shift of the pixel i, in columns across the image,
C is the number of columns in the image,
$\Delta\theta$ is the change in the vehicles heading,
Δl is the change in the vehicles lateral position,
ϕ is one half the field of view angle,
L is the distance from the vehicle's pivot point to the laser sensor,
I_i is the intensity of pixel i,
and K is the constant factor for converting pixel intensity to the distance to the corresponding point in the scene.

The right image in Figure 5.3 illustrates the results of transforming the apparent vehicle position using this equation to accurately determine how far each pixel should be shifted in the image. Notice that the trees are no longer skewed diagonally. This more realistic technique for augmenting the training set with new range image patterns allows ALVINN to learn interesting new behaviors.

5.2.3 Contour Following Using Laser Range Images

One situation ALVINN was trained to handle using laser range input was to follow terrain contours. In this domain, a network was trained using range images as a person drives the vehicle, maintaining a fixed relationship between the vehicle and objects in the environment. For instance, ALVINN was trained to drive down a street lined with parked cars, maintaining a certain lateral distance between the robot vehicle and the parked cars. If a car was protruding farther into the street than the rest, ALVINN learned to swerve to avoid it. Again because of the slow frame rate of the laser sensor, ALVINN was only able to safely drive at a speed of about 5 mph.

5.2.4 Obstacle Avoidance Using Laser Range Images

Laser range images were also employed to teach ALVINN to avoid obstacles. In this scenario, a network was trained to imitate the reactions of a person as he drove

through an obstacle-rich off-road environment. The human trainer was instructed to steer straight whenever possible, and to swerve to avoid any obstacles that appear in front of the vehicle in the direction that required the least deviation from a straight. The training and testing environment included natural obstacles such as trees, and also strategically placed trash cans.

After watching the person drive for about 5 minutes through this environment, the network was able to imitate his reactions. The network dictates a straight ahead steering direction unless confronted with an obstacle in its path. In that case, it swerved to avoid the obstacle with as shallow a turn as possible, and then returned to steering straight ahead.

5.3 Quantitative Performance Analysis

A very difficult aspect of conducting research in the area of autonomous navigation is quantifying a system's performance. The wide variety of driving situations and environmental conditions a robust autonomous agent is expected to handle makes it hard to measure its ability. Quantitative performance measures are important both for making comparisons with alternative approaches to autonomous navigation and also for determining the effects of improvements in technique using the same approach. This sections describes a technique I've developed for measuring driving performance, and the insights it provides.

The first question to answer is what aspect of driving is most appropriate for quantifying performance. The range of driving situations a system is capable of handling would be a worthwhile characteristic to employ, and one by which ALVINN would score very well relative to other systems (see Chapter 11 for more details). Unfortunately, measuring such a vague concept as flexibility is very difficult. What is required is an aspect of driving that is both measurable and gives a good indication of driving performance.

The difference between the steering command issued by the person when an image was taken and the steering command issued by the network on the same image is one simple measure I employ throughout this dissertation for quantifying network performance. However there are three drawbacks to this measure. First, it assumes the person is always steering in the correct direction, which, as will be seen below, is not always the case. Second, the difference between the steering commands of the network and a person only measures performance in the relatively restricted and "benign" situations encountered while a person drives.

In other words, since the human driver never makes drastic steering mistakes, this technique fails to measure a network's ability to recover from errors.

Finally, this technique only measures network static performance on isolated images. A significant implicit assumption of the feedforward architecture employed in ALVINN is that stable driving can be achieved without recurrent information from previous frames. The network receives only the current sensor image as input, and has no feedback connections to maintain internal state information. A frame-by-frame measure of agreement between the person and the network while a person drives cannot test the validity of this assumption, since it provides no information regarding the dynamic behavior of the vehicle under network control.

The only true measure of dynamic driving accuracy is to measure the vehicle's deviation from a known correct trajectory. For road following, the domain most easily measured, this entails measuring the vehicle's lateral displacement relative to the road center (or the center of its lane). To make this measurement and determine ALVINN's driving accuracy, I developed the following technique. First, a site for the test needs to be chosen. I chose the single lane paved road described above because of the diversity in its appearance and its infrequent use by other vehicles. Its lack of traffic facilitated the next step, which involves precisely determining the position of the road center. To ensure accuracy and repeatability of road center measurements, I manually located the road edges along the test section at 1 meter intervals and marked the point midway between them as the road center. This measurement process was difficult since broken pavement and fallen leaves frequently made estimating the exact location of road edges difficult. To avoid disturbing the network with spurious image features, the road center points were marked with paint which contrasted only slightly with the pavement, and hence did not appear in the sensor input to the network.

The second step in quantifying driving accuracy was to measure how closely the network kept the vehicle to the road center while driving autonomously. Because of the low contrast between the pavement and the road center markers, the only feasible technique for determining how close to the center the network kept the vehicle was to measure it manually. Accurate real time measurement of the vehicle's relative position every meter proved impractical, since the vehicle travels too quickly. Instead, I designed an apparatus that allows the vehicle to leave a trail which could be carefully measured after the test run. Specifically, I immersed one end of a narrow tube in a bucket of water, and attached the other end to the pivot point of the vehicle at the midpoint of the rear axle. Using this apparatus as a siphon, the vehicle dripped water along the precise path it followed over the test

road. By measuring the deviation of this drip trail from the road center marks, I could determine how accurately the network drove. Since the trail was only water, it evaporated quickly. This prevented the trail from interfering with trails made on subsequent runs.

By using this technique, precise measurement of driving accuracy can is possible. The performance of networks trained using different techniques was presented in Chapter 4. There it was shown that augmenting the training set with transformed images significantly improved the driving accuracy of a network. The technique can also be used to quantitatively compare the performance of neural networks with other techniques for vehicle control. One such comparison I have made is between ALVINN's steering performance and that of a typical human driver. In this test, ALVINN was trained on a disjoint section of the test road using the training on-the-fly technique described in Chapter 4. The network was then allowed to drive over the 140 meter stretch of calibrated test road, and its deviation from the road center recorded. The human trainer was then asked to drive over the same section of test road, under the same conditions, and his driving accuracy was recorded.

Over three runs, with the network driving at 5 miles per hour along the 140 meter test section of road, the average position of the vehicle was 1.6cm right of center, with a standard deviation of 7.2cm. The average distance the vehicle was from the road center during the test runs was 6.9cm[1]. Under human control, the average position of the vehicle was 4.0cm right of center, with a variance of 5.47cm. The average distance the vehicle was from the road center while the person drove was 5.70cm. It appears that the human driver, while more consistent than the network, had an inaccurate estimate of the vehicle's centerline, and therefore drove slightly right of the road center. Studies of human driving performance have found similar steady state errors and variances in vehicle lateral position. Blaauw [Blaauw, 1982] found consistent displacements of up to 7cm were not uncommon when people drove on highways. Also for highway driving, Blaauw reports standard deviations in lateral error up to 16.6cm.

The human trainer's bias towards driving slightly to the right of the road center manifested itself as bias towards the right in the network. Surprisingly, the network's bias towards the right appeared to be slightly less than that of the

[1]The reason these network performance measurements are significantly better than those presented in Chapter 4 for a network trained on the same stretch of road using the same training technique is that the road was much more heavily obscured by fallen leaves in the test described in Chapter 4.

human trainer. However this difference was not found to be statistically significant. Unfortunately, a quantitative performance comparison between ALVINN and other autonomous navigation systems is impossible at this time since no one has carefully measured the driving accuracy of other systems using this or any other technique. Hopefully the development of the technique described above, and the challenge of bettering the performance reported here will encourage others to make quantitative measurements of their systems' driving accuracy. For a qualitative comparison of ALVINN and other autonomous navigation systems, see Chapter 11.

5.4 Discussion

In this chapter I have demonstrated the power and flexibility of the techniques developed in Chapters 2-4 for neural network based autonomous navigation. ALVINN's ability to drive in such a wide variety of situation stems from two characteristics. First, since ALVINN learns from example employing only minimally preprocessed sensor input, it can easily adapt to new situations. As will be seen in Chapter 11, this contrasts sharply with previous autonomous driving systems. In fact, this ability to cope with novel circumstances has allowed ALVINN to drive in a wider variety of situations than any other autonomous navigation system.

The second key characteristic underlying ALVINN's success is the fact that individual networks are trained to drive in only relatively specific situations. By restricting each network's task, training is faster and performance in each domain is better than if a single network was trained to drive in all circumstances.

However, in addition to accurately driving in a number of separate circumstances, a truly autonomous system must be capable of seamlessly handling transitions from one driving situation to another. With separate networks for each circumstance, coping with a variety of situations necessitates multi-network arbitration. Chapters 7 and 8 present techniques for integrating multiple networks using rule-based and connectionist techniques respectively. These techniques allow ALVINN to perform interesting hybrid behaviors such as simultaneously following the road *and* avoiding obstacles that appear in front of the vehicle.

But before exploring schemes for multi-network arbitration, an important question related to individual driving networks needs to be addressed. It is in fact the question most frequently asked about this research, namely, "how do the networks actually process the input images to determine correct response?". In

the next chapter I present techniques for analyzing the internal representations developed by multi-layered perceptrons, and then use these techniques to gain interesting insights into the processing performed by individual driving networks.

Chapter 6

Analysis of Network Representations

A criticism frequently leveled at the connectionist paradigm is that while it may work, it is of little use since we don't understand how or why it works. While I would dispute the claim that a "black-box" connectionist solution to a difficult problem is entirely useless, I agree that understanding the details of a solution is important. Understanding the processing being performed can point out situations or conditions where the system is likely to fail. Knowing a system's limitations is critical in domains such as autonomous navigation where mistakes can be catastrophic. Also, recognizing a system's limitations and the reasons underlying them are prerequisites for improving the system's performance. As a result, I have spent considerable time and effort developing techniques for analyzing a network's internal representations.

In this chapter, I describe two techniques used to gain insight into ALVINN networks and how the resulting insights have led to improvements in performance. The first technique, involving qualitative interpretation of network weight diagrams, has been widely employed by others [Cottrell, 1990, Waibel et al., 1987, LeCun et al., 1989]. But as will be seen, the importance of spatial structure in the task of autonomous navigation makes weight diagrams more useful in this domain than in others. The second network analysis technique, called hidden unit sensitivity analysis, further exploits the strong spatial regularity in the task to quantitatively measure the influence of individual hidden units on network processing in various situations.

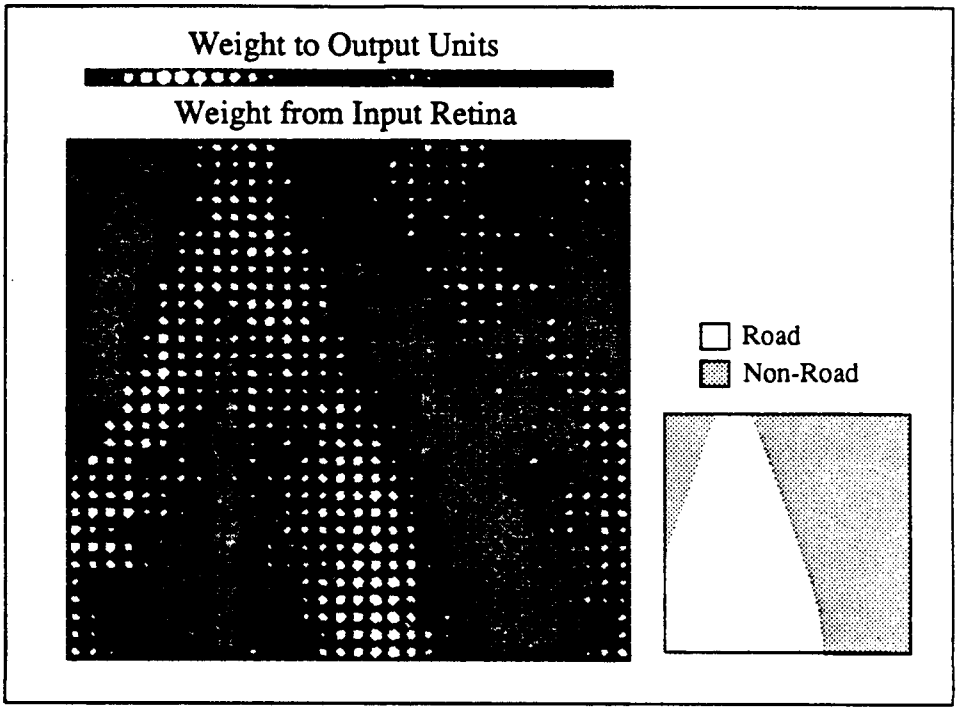


Figure 6.1: Diagram of weights projecting to and from a typical hidden unit in a network trained on roads with a fixed width. This hidden unit acts as a filter for a single road on the left side of the visual field as illustrated in the schematic.

6.1 Weight Diagram Interpretation

Weight diagrams have been introduced in Chapters 2 and 4 as a tool for inferring properties of the internal representations developed by ALVINN networks. In this section I will systematically illustrate the value of this technique by employing it to demonstrate how the flexibility across driving domains is achieved in ALVINN. In particular, I will show that the internal representation developed by the network for performing a driving task depends dramatically on the characteristics of the environment. When trained on examples of roads with a fixed width, the network develops a representation in which hidden units act as filters for roads at different positions. Figures 6.1 and 6.2 are diagrams of the connections projecting to and from single hidden units in such a network.

In Figure 6.1, the weights from the video camera retina to this hidden unit

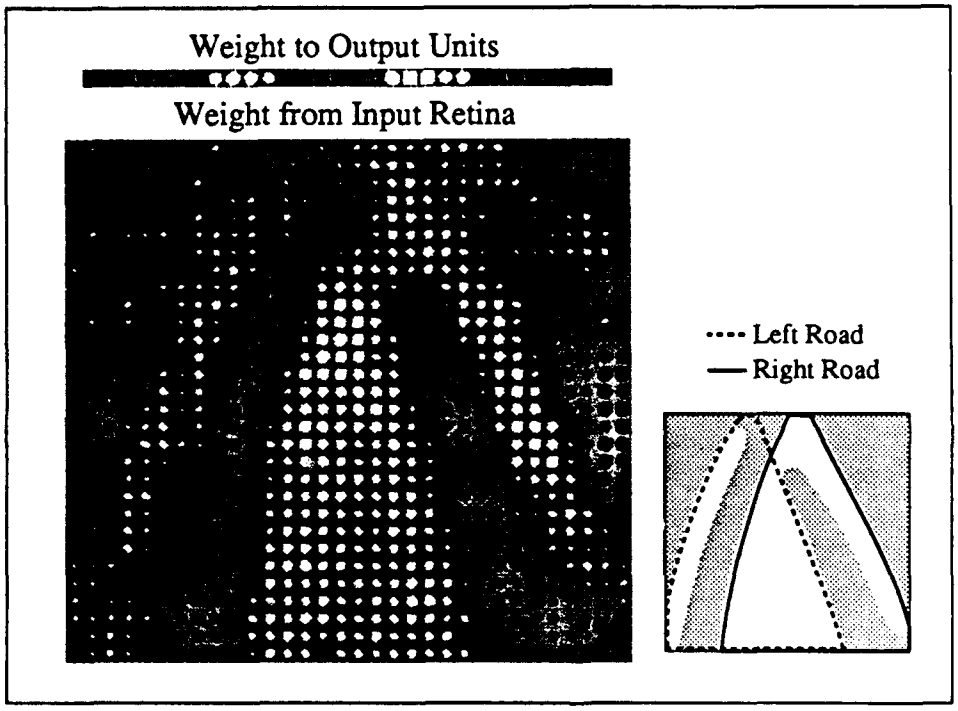


Figure 6.2: Diagram of weights projecting to and from a typical hidden unit in a network trained on roads with a fixed width. This hidden unit acts as a filter for two roads, one slightly left and one slightly right of center, as illustrated in the schematic.

support the interpretation that this hidden unit is a filter for a single road on the left side of the visual field (see the small schematic to the right of the weights from the video retina in Figure 6.1). The single road filter interpretation is reflected in the weights from this hidden unit to the direction output units. Specifically, this hidden unit makes excitatory connections to the output units on the far left, dictating a sharp left turn to bring the vehicle back to the road center. It also strongly inhibits steering in two other directions, slightly left and sharply right.

However there are interesting subtleties to this hidden unit's representation which undermine such a simple explanation. First, there are a number of inhibitory weights from the area of the input retina which represent the middle of the road according to the above explanation. These weights demonstrate that this hidden unit is actually combining two classical image processing techniques, region finding and edge detection, to locate the road. This hidden unit has learned to detect a trapezoidal-shaped region on the left of the image, but with special emphasis placed on finding the trapezoid's border. In other words, it has learned that roads resemble a trapezoid, but that the trapezoid's center is much less important than its edges for determining the correct response. The ability of connectionist networks to subtly combine processing strategies in this manner gives them a greater flexibility than hand-coded algorithms. This advantage will be discussed more in Chapter 11.

There is another subtle aspect of this hidden unit's representation which undermines the simple, single road explanation for its processing. Specifically, there are a number of excitatory connections from the *right* side of the input retina and there are three slightly excitatory connections to the output vector suggesting a slightly right turn. In fact, this hidden unit is marginally excited by a road slightly right of center, and it weakly suggests a slight right turn.

A hidden unit which clearly responds to roads at more than one position is shown in Figure 6.2. This hidden unit also acts as a filter for two roads, one slightly left and one slightly right of center. The weights from the video camera retina along with the explanatory schematic in Figure 6.2 show the positions and orientations of the two roads. This hidden unit makes bimodal excitatory connections to the direction output units, dictating a slight left or slight right turn. Hidden units which act as filters for one to three roads are the representation structures most commonly developed when the network is trained on roads with a fixed width.

The network develops a very different representation when trained on images with widely varying road widths. A typical hidden unit from this type of representation is depicted in Figure 6.3.

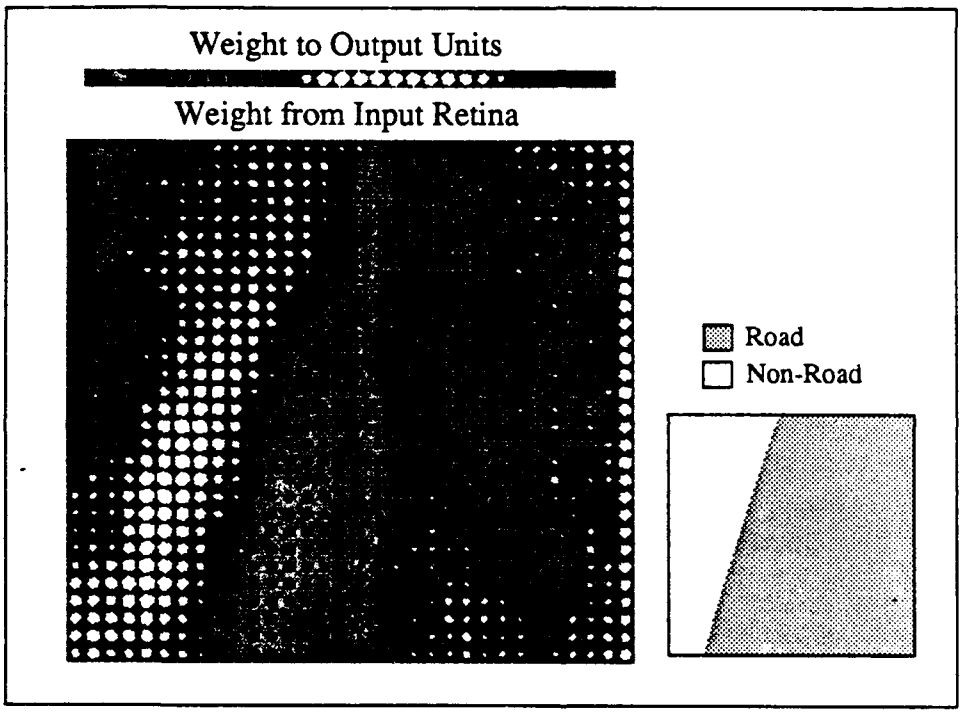


Figure 6.3: Diagram of weights projecting to and from a typical hidden unit in a network trained on a road with varying width. This hidden unit acts as a detector for the left road edge, as illustrated by the schematic.

The first important feature to notice in Figure 6.3 is the single abrupt change in weights along the diagonal line on the left side of the weights from the video camera retina to this hidden unit. This edge clearly indicates that this hidden unit is a filter for the left edge of the road (see the schematic to the right of the video retina weights). In addition, notice that unlike in the earlier weight diagrams, the weights from the units which are supposed to represent the road are inhibitory (black) rather than excitatory (white). This pattern indicates that this hidden unit responds to roads which are darker, and hence have lower retina activation levels than the non-road. This illustrates that when the training set requires it, the network will develop units to deal with roads which can be either darker or lighter than the non-road.

The hidden unit in Figure 6.3 supports a rather wide range of travel directions, as indicated by the large number of excitatory connections it makes to the output layer. This is to be expected, since the correct travel direction for a road with an edge at a particular location varies substantially depending on the road's width. This hidden unit cooperates with hidden units that detect the right road edge to determine the correct travel direction in any particular situation. The large magnitude weights coming from the image column on the far right are puzzling and do not quite fit the interpretation I have provided. This anomaly demonstrates that subjective weight diagram interpretation can only provide a gross explanation of the processing performed by individual hidden units. In the remainder of chapter I present a technique called sensitivity analysis that can be used to understand the more subtle aspects of a network's representation.

6.2 Sensitivity Analysis

While qualitative hidden unit analysis can provide useful insights into the representations developed by trained networks, it has two major limitations. First, hidden unit interpretation is not always as straightforward as in the above examples. Consider the hidden unit depicted in Figure 6.4. It is clear from the structure in the input-to-hidden and hidden-to-output weights that this unit acts as some type of feature detector, but exactly how it contributes to road following is not immediately evident. In addition, weight diagrams such as this can't shed light on how this hidden unit works together with other hidden units to produce the correct output response.

To quantitatively understand the processing performed by individual hidden

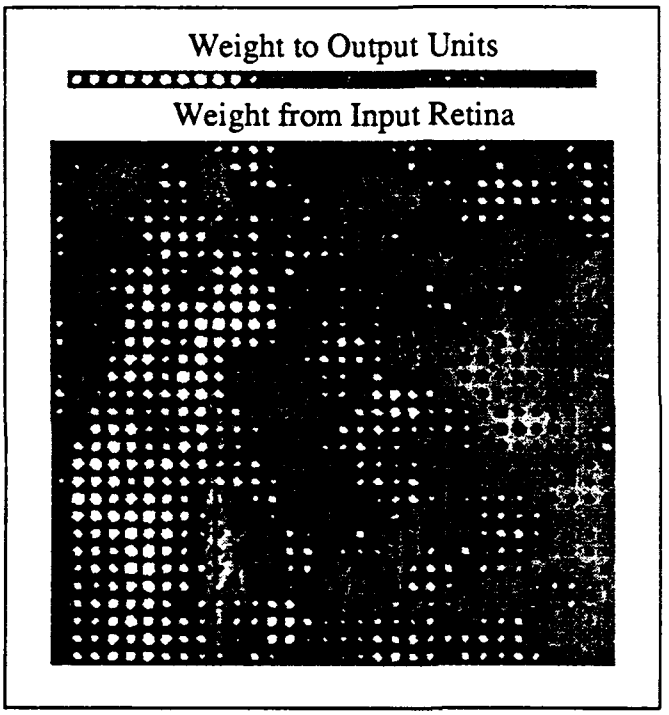


Figure 6.4: Weight display for a more typical, less obvious hidden unit from a network trained on single lane road images.

units, and the interaction among multiple hidden units, I've developed a technique called hidden unit sensitivity analysis. This technique is a more rigorous implementation of a procedure frequently used in classical neuroscience investigations of biological visual systems [Hubel & Wiesel, 1979]. The neuroscience procedure involves systematically presenting the retina with a variety of stimuli to determine the situations which cause an internal neuron to respond. The simplicity of three layer artificial neural networks allows us to go further. Using sensitivity analysis it is possible to determine not only the situations which simulate a hidden unit, but also the impact the hidden unit has in those situations.

6.2.1 Single Unit Sensitivity Analysis

From the positive weights on the left side of the hidden unit's receptive field in Figure 6.4, it appears that the hidden unit will respond when the road is on the left side of the image. In addition, from the positive weights on the left side of the hidden unit's "projective field" (i.e. the weights from the hidden unit to the output units [Lehky & Sejnowski, 1988]), it appears that this unit will dictate a left turn when it has a positive activation value.

The first step towards achieving a quantitative understanding of this hidden unit's processing with sensitivity analysis is to systematically vary the position and orientation of the road in images presented as input to the network, and record the activation of the hidden unit for each. The result of this process is shown in Figure 6.5. In this diagram, the height of each point on the surface represents the activation level of the hidden unit when presented with a road image at a different position and orientation. At points above the midline, the hidden unit had a positive activation level, while points below mean the hidden unit had a negative activation level[1]. As can be seen from this diagram, the hidden unit responds with a positive activation level when presented with images in which the road is situated towards the left side and/or the heading is towards the left. Conversely, roads on the right side and/or heading towards the right produce a negative response from this hidden unit.

Figure 6.6 was also made by systematically varying the position and orientation of roads in the input image, but this time the height of the surface represents the influence a positive activation level in this hidden unit will have on the correct steering direction output unit for this case. In other words, for each road position

[1]Recall that ALVINN employs a symmetric sigmoid activation function ranging from -1 to +1.

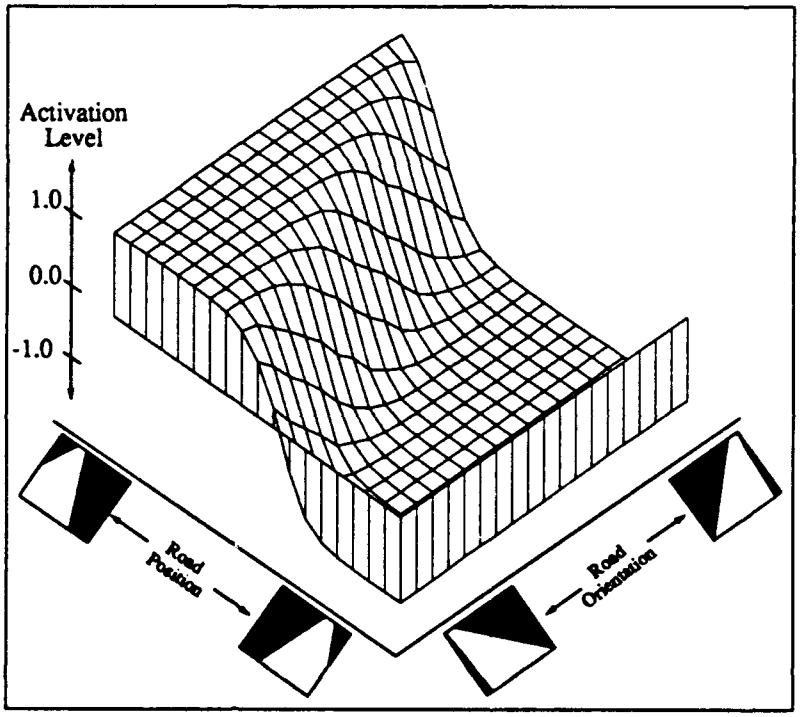


Figure 6.5: Response of a single hidden unit in an ALVINN network to images with roads at differing positions and orientations.

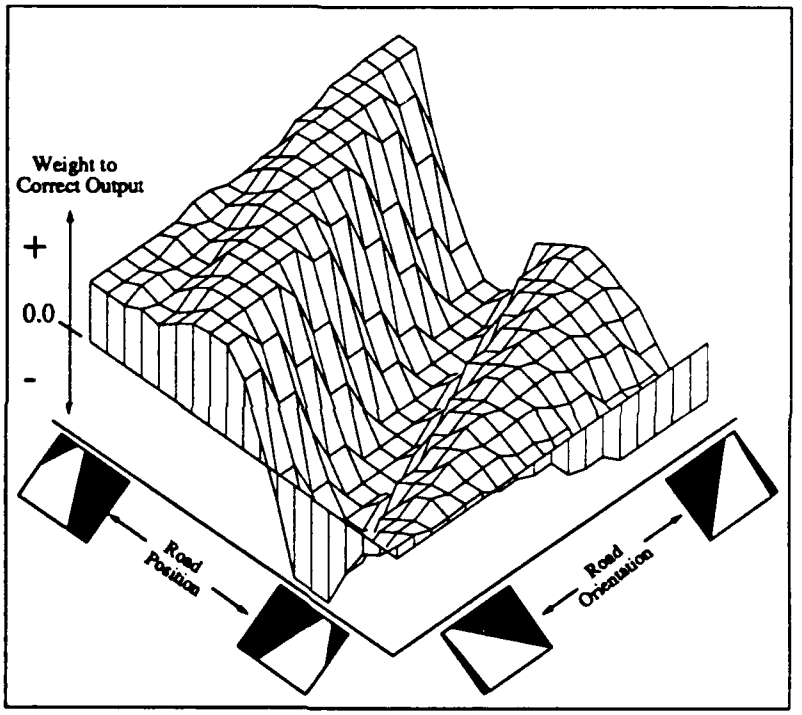


Figure 6.6: Weight from the hidden unit to the output unit representing the correct steering direction for roads at differing positions and orientations.

and orientation, there is a single most correct steering direction that is represented by a unit in the output layer of the network. This graph represents the weight from the hidden unit to that correct output unit for each road position and orientation. The positive influence from the hidden unit to the left turn output units illustrated in the weight diagram of Figure 6.4 is also evident in this figure as the elevated plateau along the back left of the surface. If the hidden unit had a positive activation level, it would excite the correct output unit when the road is on the left side of the image since it has positive weights to the left turn output units.

The diagonal "valley" down the middle of the diagram illustrates that a positive activation level for this hidden unit would result in inhibition of the correct output in cases when the road is near the middle of the image and heading nearly straight ahead. The diagonal tendency of features in this and the other sensitivity analysis diagrams results from the fact that for the purpose of steering correctly, road images are divided into equivalence classes. The correct steering direction is the same for a road centered in the image heading towards the left and for a road on the left of the image heading straight ahead.

Finally Figure 6.7 was constructed by multiplying Figures 6.5 and 6.6 together. Multiplying the graph of the hidden unit's response to various road images and the graph of the hidden unit's potential contribution on various road images results in a graph showing the contribution the hidden unit actually makes to the correct output unit for the various road position/orientation combinations. Multiplying the a unit's state by its outgoing weights to determine its influence is analogous to "weight-state" representation developed by Gorman and Sejnowski [Gorman & Sejnowski, 1988]. While Gorman and Sejnowski used a multiplicative method to determine the influence of inputs on individual hidden units, sensitivity analysis uses products to determine the effect hidden units have on a very special output unit, the correct one.

As was expected from the weight diagram interpretation, the hidden unit makes a significant positive contribution to activating the correct output unit for roads on the left of the image and heading towards the left. This utility for left turn situations is illustrated by the plateau along the back left of the surface in Figure 6.7.

In addition, the diagonal ridge down the center of the surface demonstrates a less obvious contribution of this hidden unit. It shows that on roads where the correct response is to steer straight ahead, the negative activation level of the hidden unit when coupled with the inhibitory weight between it and the straight ahead output units results in support for steering straight ahead. This illustrates an advantage of using a symmetric activation function; namely that hidden units

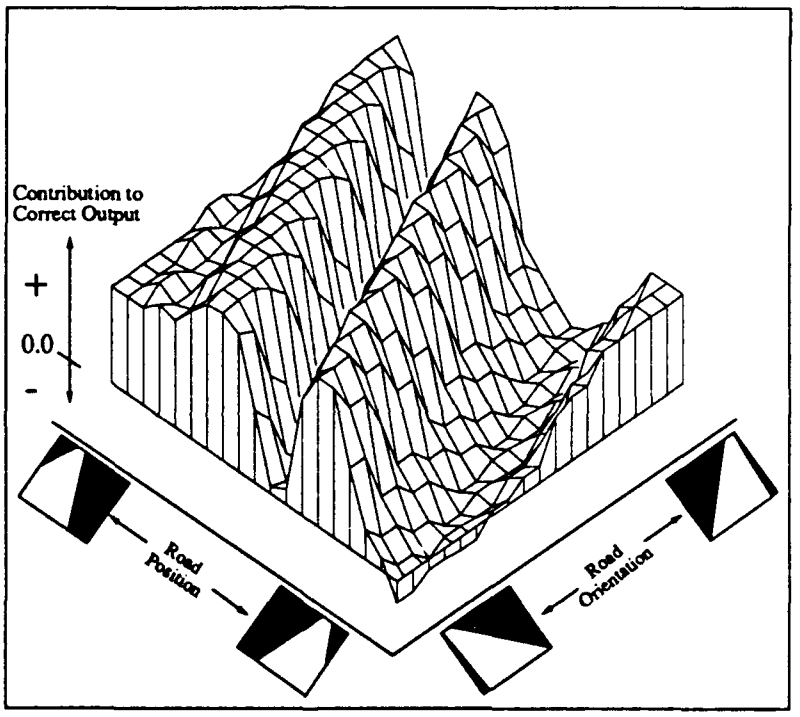


Figure 6.7: Contribution of the hidden unit to the correct steering direction unit for roads at differing positions and orientations.

can make significant contribution to the network's at either output extreme. This "two negatives makes a positive" effect is also responsible for the somewhat small positive influence this unit has on roads requiring an extreme right turn, depicted by the smaller peak in Figure 6.7 in the far right corner.

Another interesting feature illustrated by Figure 6.7 is that the hidden unit makes significant positive contributions to the correct output unit on a large fraction of the possible road images. Also, even when this hidden unit is not making a strong positive impact on the correct output unit, it almost never has a detrimental influence on the network's processing. This is illustrated in Figure 6.7 by the fact that on only a few images does the surface fall below the plane representing zero contribution to the correct output unit.

6.2.2 Whole Network Sensitivity Analysis

To fill in the "gaps" left by a hidden unit and counteract its small negative impact on the correct output unit in a few cases, the other hidden units must also contribute. To illustrate this dovetailing of coverage by multiple hidden units, consider Figures 6.8 and 6.9. Figure 6.8 depicts the weight diagram for a network trained on a single lane road. It is difficult to determine from visual inspection of this display how the hidden units work together to process the range of possible road images they might encounter.

But when the contributions to the correct output unit are displayed together, as in Figure 6.9, the coordination between multiple hidden units becomes clear. In this diagram, the top four graphs represent the contributions from each of the four hidden units in the network from Figure 6.8 to the correct output unit. The hidden units work together to "fill in the holes" in each other's coverage, so that the net contribution to the correct output unit is strongly positive for all road positions and orientations (although the total contribution to the correct output unit on sharp left turn images is somewhat less than on the rest). This is shown in the total contribution graph at the bottom left of Figure 6.9. This diagram, which shows the total input to the correct output unit for various road positions and orientations, was constructed by summing the hidden units' individual contributions depicted in the top four graphs. The lower right graph in Figure 6.9 shows the activation level of the correct output unit. It was created by applying the sigmoid squashing function to the net input graph in the lower left.

An interesting property of the network illustrated by Figure 6.9 is that only two of the four hidden units (hidden units 1 and 3) are responsible for turning on

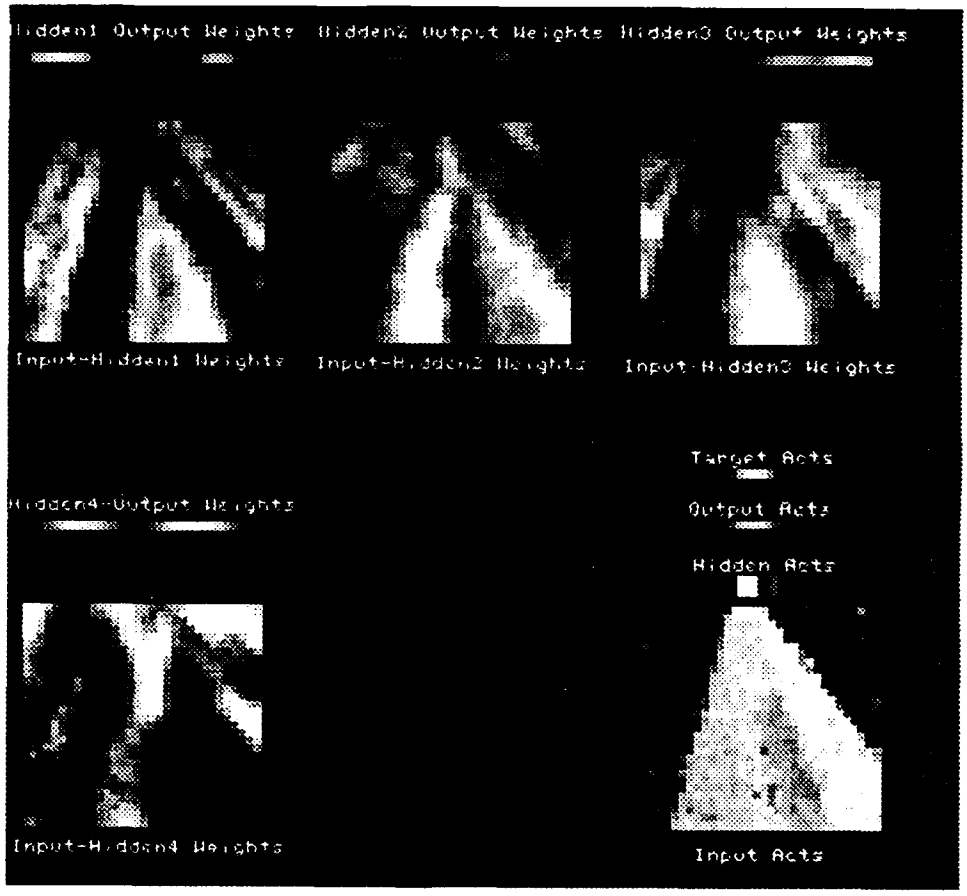


Figure 6.8: Weight displays for all the hidden units from a network trained on single lane road images.

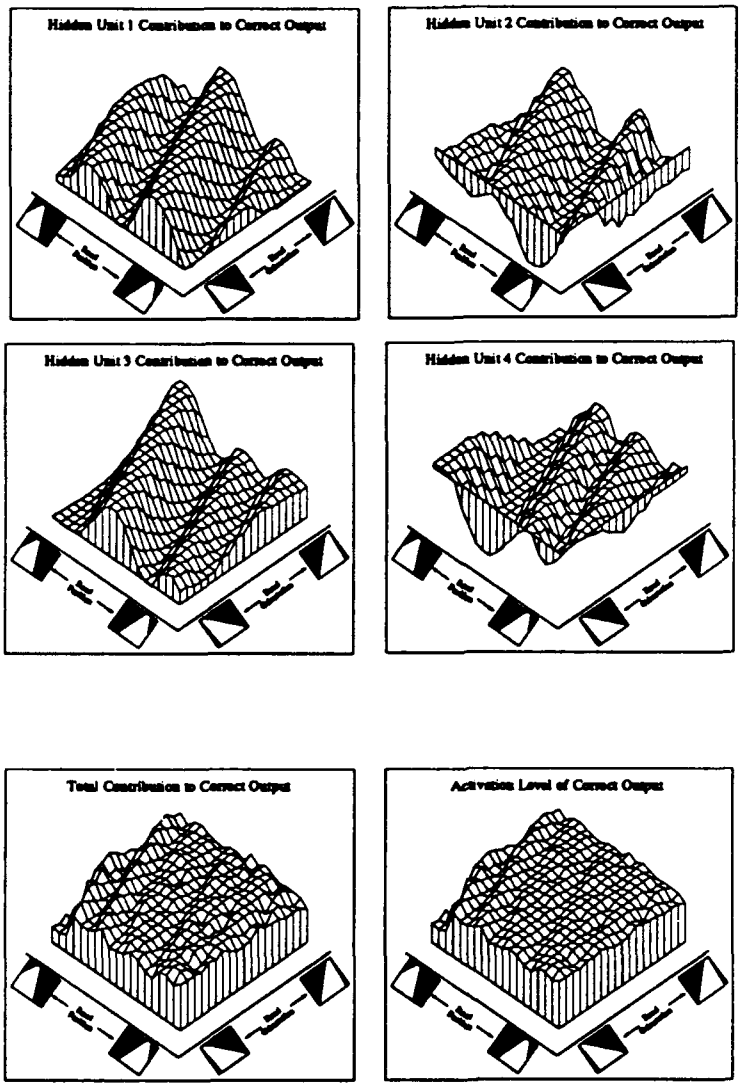


Figure 6.9: The upper four graphs represent the contribution from each of the four hidden units to the correct output unit for different road positions and orientations. The graph at the lower left represents the summation of all the hidden unit contributions to the correct output unit. The graph at the lower right represents the activation level of the correct output unit as road position and orientation is varied. Notice the activation level of the correct output unit are positive for all road position/orientation combinations.

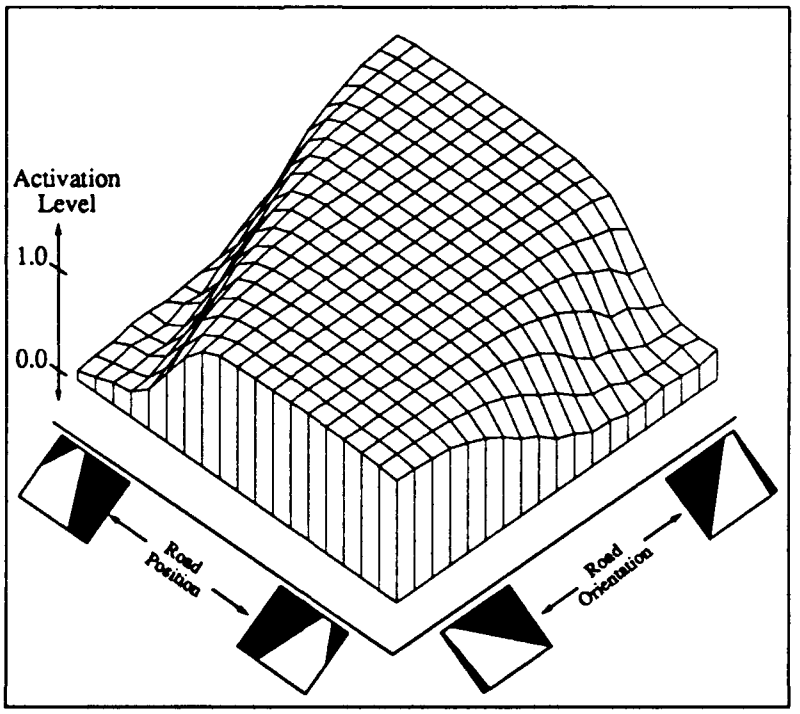


Figure 6.10: Response of hidden unit 2 to images with roads at differing positions and orientations.

the correct output unit. Hidden units 2 and 4 have an *inhibitory* influence on the correct output unit for almost all road positions and orientations. The question is, if hidden units 2 and 4 are not helping turn on the correct output unit, what are they doing? They must be contributing in some way towards minimizing the network's error, since if they weren't, the forces of gradient descent and weight decay would reduce their detrimental influence, and hence the network's total error, by zeroing the weights from these hidden units to the output units.

The activity pattern of one of these seemingly detrimental hidden units does not provide much insight into its function. Figure 6.10 shows the activation level of hidden unit 2 upon presentation of roads at different positions and orientations. This graph illustrates that hidden unit 2 is excited by a wide range of road stimuli, but it tells us little else.

To really understand what these seemingly detrimental hidden units are doing, the first step is to realize that turning on the correct output unit is not the only task the network must perform. In addition, it must inhibit incorrect output units to

prevent them from being chosen as the direction in which to steer. It is for this task that hidden units 2 and 4 are specializing, as is evident from Figure 6.11. This diagram shows the contribution from each of the hidden units and the network as a whole not to the *correct* output unit, but to one of the *incorrect* output units. Specifically, the four graphs at the top of Figure 6.11 represent the contributions from the four hidden units to the output unit that is 15 output units away from the correct one along the 30 unit output vector. As can be seen from this diagram, hidden units 2 and 4 are crucial for inhibiting this incorrect unit, particularly on the road images requiring a shallow turn, which the other two hidden units don't handle. As a result of their large inhibitory influence, the net input to the incorrect output unit is negative across all road positions and orientations, as shown in the bottom left graph of Figure 6.11. After applying the sigmoid squashing function to this net input, the activation level for this incorrect output unit is very close to -1 for all road position/orientation combinations, as shown in the bottom right graph of Figure 6.11.

As is evident from Figures 6.9 and 6.11, this network's performance is almost perfect, since it turns on the correct output unit and turns off (at least one) incorrect output unit across all road positions and orientations. However ALVINN networks do not always achieve such proficient performance on all driving situations. Figure 6.12 shows a weight diagram of a network trained on a stretch of single lane road containing only straight stretches and right turns. For this test, no transformations were performed to increase training set variety, so the network never saw any left turns. From the weight diagram it is difficult to infer that there is anything wrong with the network. However the activation level diagram for the network depicted in Figure 6.13 clearly illustrates the network's shortcomings. This diagram shows that on roads requiring a left turn, this network strongly inhibits the correct steering direction unit, and hence will perform poorly.

Figures 6.14 and 6.15 show the weight diagram and total contribution diagram for an even less reliable network. This network was trained on a stretch of road without left or right turns. Again, no transformations were used to aid the network's generalization. As a result, the network is incapable of correctly responding to road images requiring sharp turns either to the left or right. Again, it would be difficult to draw this conclusion directly from the weight diagram, but the diagram representing the activation level of the correct output unit makes this fact apparent.

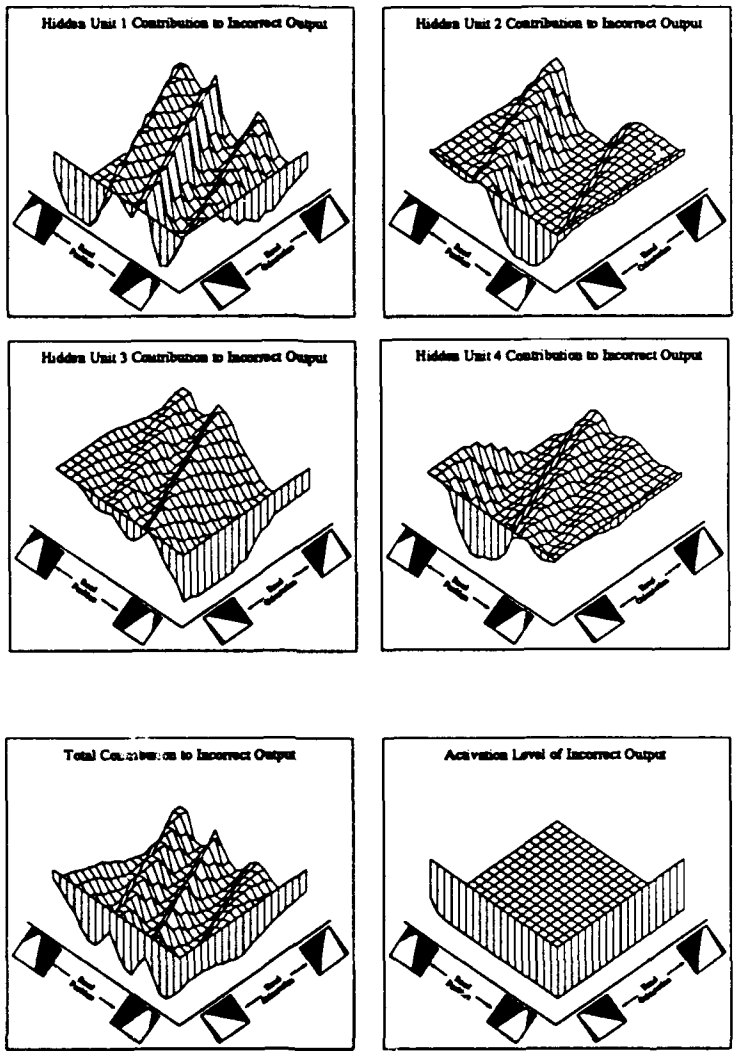


Figure 6.11: The upper four graphs represent the contribution from each of the four hidden units to an incorrect output unit for different road positions and orientations. The graph at the lower left represents the summation of all the hidden unit contributions to the correct output unit for different road positions and orientations. The graph at the lower right represents the activation level of an incorrect output unit as road position and orientation are varied. Notice that the activation level of the incorrect output unit is negative for all road position/orientation combinations.

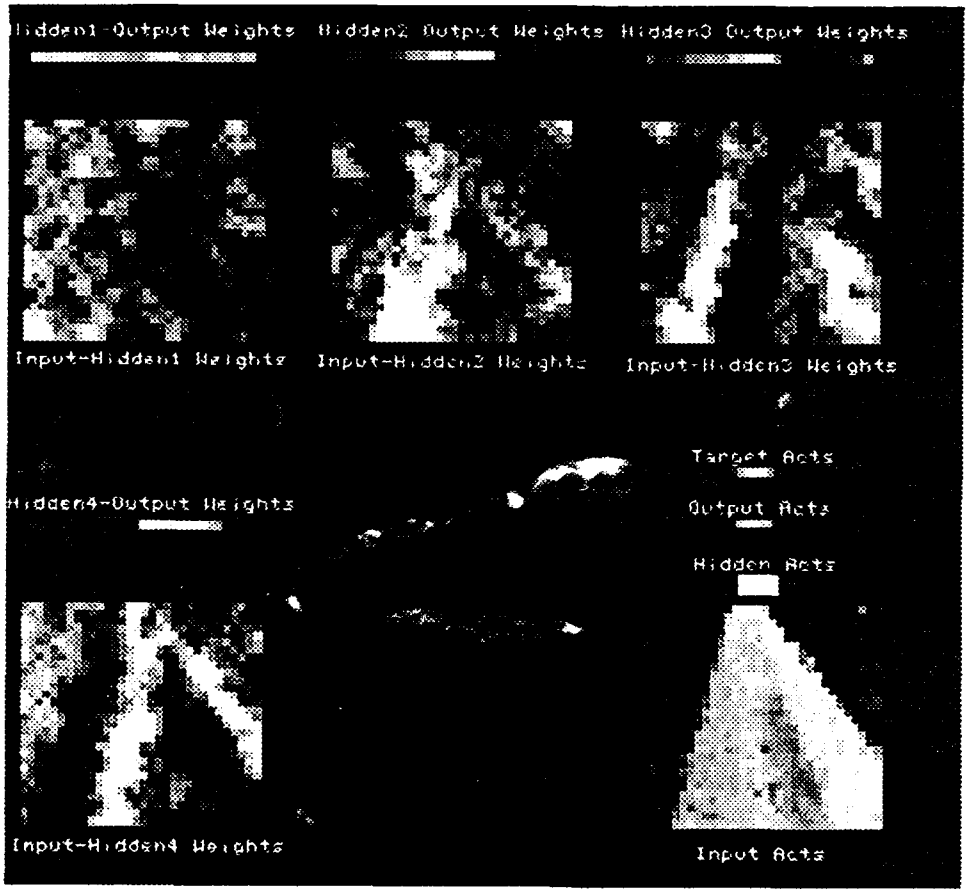


Figure 6.12: Weight diagram from a network trained on a stretch of single lane road images containing no left turns.

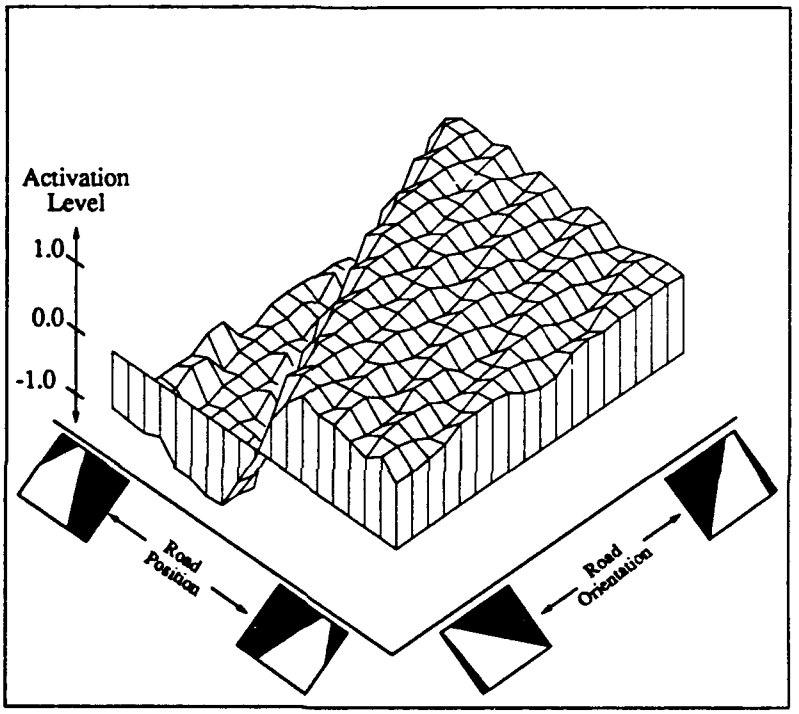


Figure 6.13: The activation level of the correct output unit for a variety of road positions and orientations in a network trained on a stretch of road with no left turns.

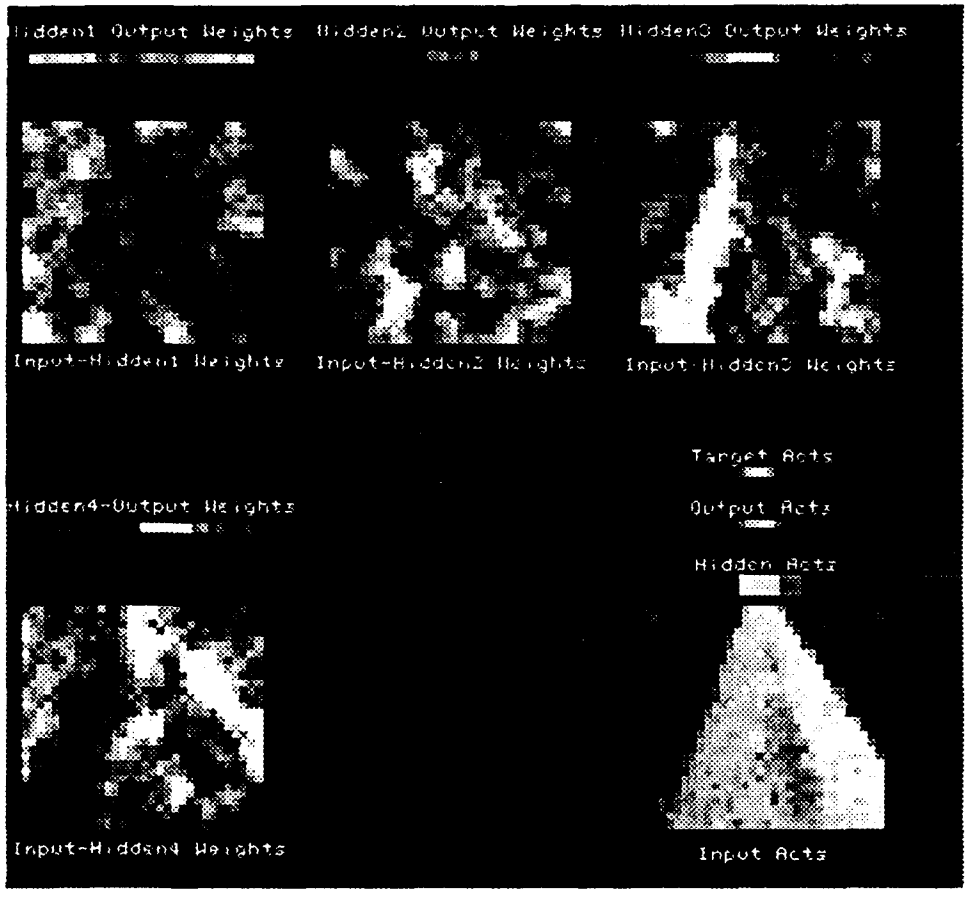


Figure 6.14: Weight diagram from a network trained on a stretch of single lane road images containing no left or right turns.

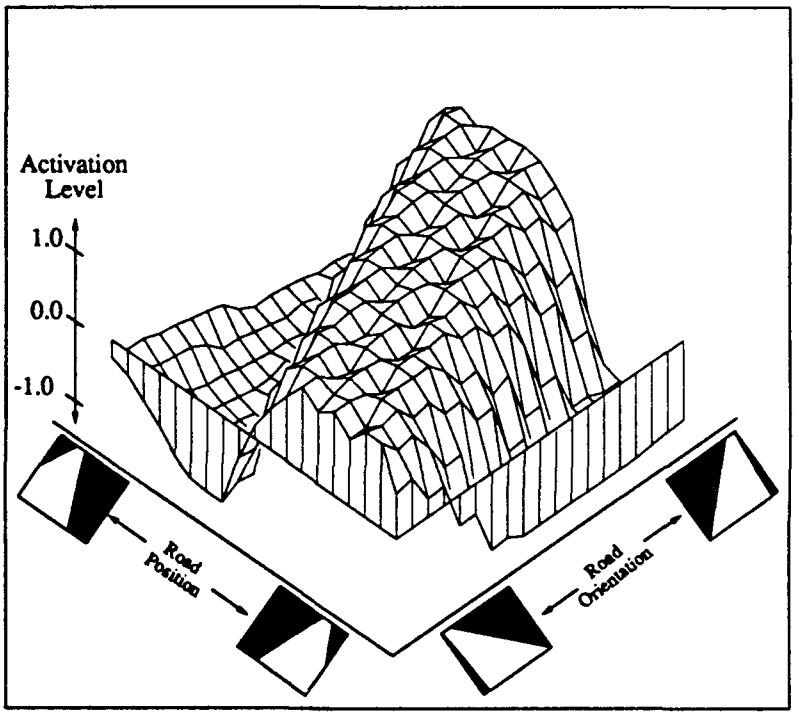


Figure 6.15: The activation level of the correct output unit for a variety of road positions and orientations in a network trained on a stretch of road with no left or right turns.

6.3 Discussion

Hidden unit sensitivity analysis allows us to go beyond the usual qualitative speculation about a hidden unit's "job" to determine empirically how a hidden unit contributes to performing the task across a range of input situations. In addition to providing insight into the behavior of isolated hidden units, sensitivity analysis can illustrate the subtle cooperation between multiple hidden units which is required for complex tasks. Finally, sensitivity analysis is yet another way of quantifying network performance. It can point out situations in which a network is likely to fail; these situations need greater representation in the training set.

Previous research in neuroscience [Hubel & Wiesel, 1979] and connectionist modeling [Lehky & Sejnowski, 1988, Zipser & Anderson, 1988] has demonstrated the utility of measuring the response of internal units to various stimuli. However this technique is not always sufficient to allow one to determine the a unit's function. This was demonstrated in Figure 6.10, in which a hidden unit whose function is to inhibit incorrect output units in a particular set of situations was shown to respond strongly in a wide range of stimuli.

Lehky and Sejnowski [Lehky & Sejnowski, 1988] were on the right track when they recognized the importance of both a unit's receptive field, and also the connections it makes to subsequent stages of processing, which they called its "projective field". However instead of subjectively interpreting a hidden unit's function from its receptive and projective fields, sensitivity analysis allows quantitative measurement of a unit's contribution to the network's processing.

But sensitivity analysis does have limitations. To be effective, it requires that the task have important continuous dimensions along which the input can be smoothly varied. For the task of autonomous driving, two obvious continuous input features are road position and orientation. Finding useful dimensions in other domains to employ in sensitivity analysis may prove more difficult. For domains such as speech and character recognition, the network must often learn to ignore the absolute spatial position of features in the input signal, and instead employ the relative position of features to correctly classify the input. For speech recognition, it may be possible to use sensitivity analysis by shifting the frequency or the type scale of input signal and recording the contributions of hidden units.

For classification tasks such as speech and character recognition, the technique of cluster analysis [Gorman & Sejnowski, 1988, Servan-Schreiber et al., 1989], which groups input patterns according to the similarity of their corresponding hidden unit activation patterns, is probably a more informative method for analyzing

internal representations. However there is the potential for useful cooperation between the two techniques. Analysis of the common feature(s) among the members of a cluster could be used to hypothesize important input feature dimensions. The actual importance of this feature could then be confirmed or denied by presenting the network with inputs in which this feature is continuously varied while the response and output contribution of individual hidden units are recorded.

To make the potential synergy between cluster analysis and sensitivity analysis more concrete, consider the following hypothetical example from the domain of character recognition. Suppose cluster analysis of the networks internal representation grouped the characters b, d and h into one cluster and the characters f, g, p, q, t, and y into another. It might be hypothesized from these groupings that the network is employing a character's vertical "center of mass" to perform classification. This hypothesis could be tested by presenting the network with inputs having various centers of mass and recording the responses of the hidden units. Finding hidden units whose response varies coherently with the input's center of mass would strongly suggest this as an important feature employed by the network. The input used to test the hypothesis could be either real characters distorted to alter their center of mass, or "nonsense characters" with varying centers of mass fabricated specifically for the test. If the input's center of mass is an important feature for the network, then either type of test stimulus should elicit responses from at least one hidden unit which correlates with the input's center of mass. While this example is purely fabricated, it illustrates the potential application of sensitivity analysis outside the domain of autonomous navigation.

Chapter 7

Rule-Based Multi-network Arbitration

This chapter describes work in combining artificial neural network techniques for autonomous driving with symbolic processing methods in order to achieve more intelligent behavior. This work was done in collaboration with Jay Gowdy and Charles Thorpe [Pomerleau et al., 1991d].

Artificial neural networks are commonly employed as monolithic non-linear classifiers. The technique, often used in domains such as speech, character and target recognition, is to train a single network to classify input patterns by showing it many examples from numerous classes. The mapping function from inputs to outputs in these classification tasks can be extremely complex, resulting in slow learning and unintelligible internal representations.

However there is an alternative to this monolithic network approach. By training multiple networks on different aspects of the task, each can learn relatively quickly to become an expert in its sub-domain. The training on-the-fly technique described in Chapter 3 makes this specialized expert network approach very effective for autonomous navigation. As was seen in Chapter 5, training on-the-fly allows specialized networks to be trained in under five minutes to perform a variety of navigation tasks including single-lane road driving, multi-lane highway driving, and collision avoidance.

But achieving full autonomy requires not only the ability to train individual expert networks, but also the ability to integrate their responses. This chapter focuses on rule-based arbitration techniques for combining multiple driving experts into a system that is capable of guiding a vehicle in a variety of circumstances.

These techniques are compared with other neural network integration schemes and shown to have a distinct advantage in domains where symbolic knowledge and techniques can be employed in the arbitration process.

7.1 Symbolic Knowledge and Reasoning

Despite the variety of capabilities exhibited by individual driving networks, the ALVINN system described thus far does not achieve true autonomy. The network ALVINN architecture is capable of driving only on the type of road on which it was trained. If the road characteristics changed, ALVINN would often become confused and stray from the road. In addition, a real autonomous system needs to be capable of planning and traversing a route to a goal. The neural network driving modules are good at reactive tasks such as road following and obstacle avoidance, but the networks have a limited capability for the symbolic tasks necessary for an autonomous mission. The system of networks cannot decide to turn left at an intersection in order to reach a goal. After making a turn from a one-lane road to a two-lane road, the system does not know that it should stop listening to one network and start listening to another. Just as a human needs symbolic reasoning to guide reactive processes, the networks need a source of symbolic knowledge to plan and execute a mission.

Ideally, the symbolic knowledge source would reason like a person. It would use its knowledge of the world to plan a sequence of observations and corresponding actions to traverse the route. For instance, to achieve the goal of reaching a friend's house, the mission description might be a sequence like "Drive until the sign for Seneca Road is seen, and turn left at that intersection. Then drive until the third house on the left is seen, and stop in front of it."

In this ideal system, once the mission is planned, the symbolic knowledge source would rely entirely on perception to control the execution of the mission. In other words, the symbolic resource module would be able to recognize events and use what it sees to guide the interaction of the networks. The symbolic resource module would be capable of reading the street sign at an intersection and commanding the appropriate turn to continue on to its destination. It would also be able to identify the new road type and choose the appropriate network for driving on that kind of road. Unfortunately, the perception capabilities required by such a module are beyond the current state of the art.

In order to bridge the gap between mission requirements and perception ca-

pabilities, ALVINN uses additional geometric and symbolic information stored in an "annotated map" [Thorpe & Gowdy, 1990]. An annotated map is a two dimensional data structure containing information about the area to be traversed. Each entry in the annotated map's two dimensional array covers a small patch of the environment, usually a couple meters square. Each entry is actually a linked list whose elements represent all the objects that occupy the corresponding patch of the environment. Each object description contains the type of the object, such as road, obstacle or mailbox, and the objects precise location and shape. In addition, each object in the map can be annotated with extra information to be interpreted by the clients that access the map. For example, as far as the annotated map is concerned, a mailbox is simply a two dimensional polygon at a particular location with some extra bits associated with it. The "extra bits" might represent the three dimensional shape of the mailbox, or even the name of the person who owns it. The module which manages the annotated map does not interpret this extra information, but rather provides a mechanism for client modules to access the annotations. This reduces the knowledge bottleneck that can develop in large, completely centralized systems.

The annotated map is not just a passive geometric database, but instead is an active part of our system. Besides having a 2D representation of the physical objects in a region, annotated maps can contain what are called alarms. Alarms are conceptual objects in the map, and can be lines, circles, or regions. Each alarm is annotated with a list of client modules to notify and the information to send to each when the alarm is triggered. When the annotated map manager notices that the vehicle is crossing an alarm on the map, it sends the information to the pertinent modules. Once again, the map manager does not interpret the information: that is up to the client modules.

Alarms can be thought of as positionally-based production rules. Instead of using perception-based production rules like "If A is observed, then perform action B", an annotated map based system has rules of the form, "If location A is reached, then perform action B". Thus the problem of making high level decisions from the difficult task of perceiving and reacting to external events is reduced to the relatively simple task of monitoring and updating the vehicle's position.

The first step in building an annotated map is collecting geometric information about the environment. Gowdy [Thorpe & Gowdy, 1990] has created a map building system that records the vehicle's trajectory as a person drives and inserts entries into the annotated map representing the roads location at one meter intervals. At the same time, the laser rangefinder described in Chapter 5 is used to record the

positions of landmarks such as mailboxes and telephone poles. Specifically, the range images provided by the laser sensor are converted into a three dimensional aerial map of the terrain directly in front of the vehicle using an algorithm described in [Thorpe et al., 1991]. A landmark entry is created in the annotated map representing the position and shape of each 3D object detected in the aerial map.

After encoding the terrain's geometric description in the annotated map, the map can be used to plan a particular mission. The planning phase involves adding specific instructions to the map in the form of "trigger annotations". This is currently a process performed by the person planning the mission. For example, the human expert knows that when approaching an intersection, the vehicle should slow down, so the expert chooses the appropriate location to put the trigger line. The trigger line goes across the road at that point, and is annotated with a string of bits that represents the new speed of the vehicle. During the run, when the vehicle crosses the trigger line, the map manager sends the string of bits to a module that interprets the information and slows the vehicle to the desired speed. In the current system, alarms are interpreted as commands, but there is no predefined "correct" way for a module to react to an alarm. Depending on its content, an alarm could also be interpreted as a wakeup call, or even as simply advice.

Because position information is so critical to the annotated map, the system uses multiple techniques to determine the vehicle's current location. First, an Inertial Navigation System (INS) is employed to determine the vehicle's location with an error of approximately 1% of distance traveled [Amidi & Thorpe, 1990]. To eliminate positioning error that accumulates over time in the INS data, the annotated map system also uses information from perception modules. For example, since the driving networks presumably keep the vehicle on the road, lateral error in the vehicle positioning system relative to the road can be identified and eliminated. In addition, a module using the laser rangefinder compares the landmarks it sees to the landmarks collected when the map was built, and triangulates to find the vehicle's position on the map. These techniques allow perception modules to provide useful positioning information *without* requiring them to explicitly recognize and interpret particular objects such as street signs. The position corrections provided by perception modules are interpreted as a change in the transform between the location that the INS reports and the real vehicle position on the map. A separate module, called the navigator, is in charge of maintaining and distributing this position transform.

Annotated maps provide the system with the symbolic information and control knowledge necessary for a fully autonomous mission. Since the control knowledge

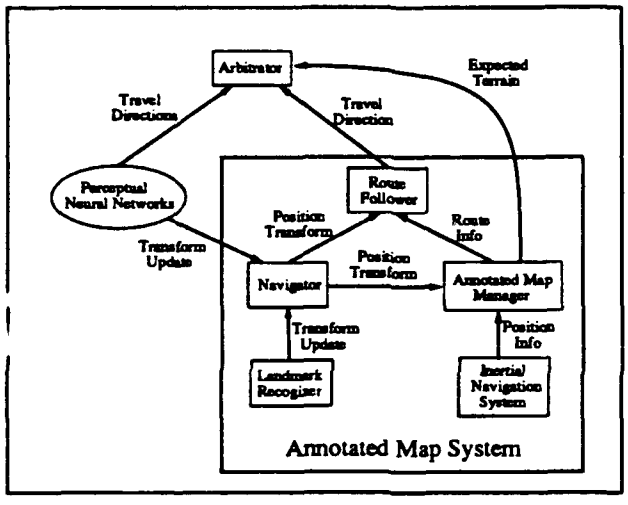


Figure 7.1: The components of the annotated map system and the interaction between them. The annotated map system keeps track of the vehicle's position on a map. It provides the arbitrator with symbolic information concerning the direction to steer to follow the preplanned route and the terrain the vehicle is currently encountering. The neural network driving modules are condensed for simplicity into a single block labeled perceptual neural networks.

is geometrically based, and since planning is done before the mission starts, runtime control comes at a low computational cost. Figure 7.1 shows the structure and interaction of the annotated map system's components. It also illustrates the annotated map system's interaction with the other parts of the system, including the perceptual neural networks and the arbitrator (discussed below). Figure 7.2 shows a map and annotations for a mission segment.

7.2 Rule-based Driving Module Integration

The symbolic knowledge provided by the annotated map system can help guide the interaction of the reactive driving networks. Figure 7.3 shows the system architecture with emphasis on the neural networks. Whereas Figure 7.1 subsumed the neural network systems into one unit labeled "perceptual neural networks", Figure 7.3 subsumes the annotated map system into one package. In this diagram, each box represents a separate process running in parallel. Images from the three onboard sensors are provided to the five driving networks shown in the second row of the diagram. The driving networks propagate activation forward through their weights, with each determining what it considers to be the correct steering direction. These steering directions are sent to the arbitrator, which has the job of deciding which network to attend to and therefore how to steer.

The arbitrator makes use of both the geometric and control information provided by the annotated map system to perform a mission autonomously. First, the route following module within the annotated map system uses the geometric information in the annotated map to recommend a vehicle steering direction. The direction recommended by the route follower is the direction it thinks the vehicle should steer in order to follow the preplanned route. When the vehicle is driving down a road, the route follower queries the annotated map for the position of the road ahead of the vehicle. The route follower uses this geometric information to generate a steering direction.

The annotated map system also provides the arbitrator with information about the current driving situation, including what type of road the vehicle is on, and whether there is an intersection or dangerous permanent obstacle ahead. For example, suppose during the planning phase the human expert notices that at a particular point the road changes from one lane to two. The expert would set a trigger line at the corresponding point on the map and annotate it with a message that will tell the arbitrator to stop listening to the one-lane road following network

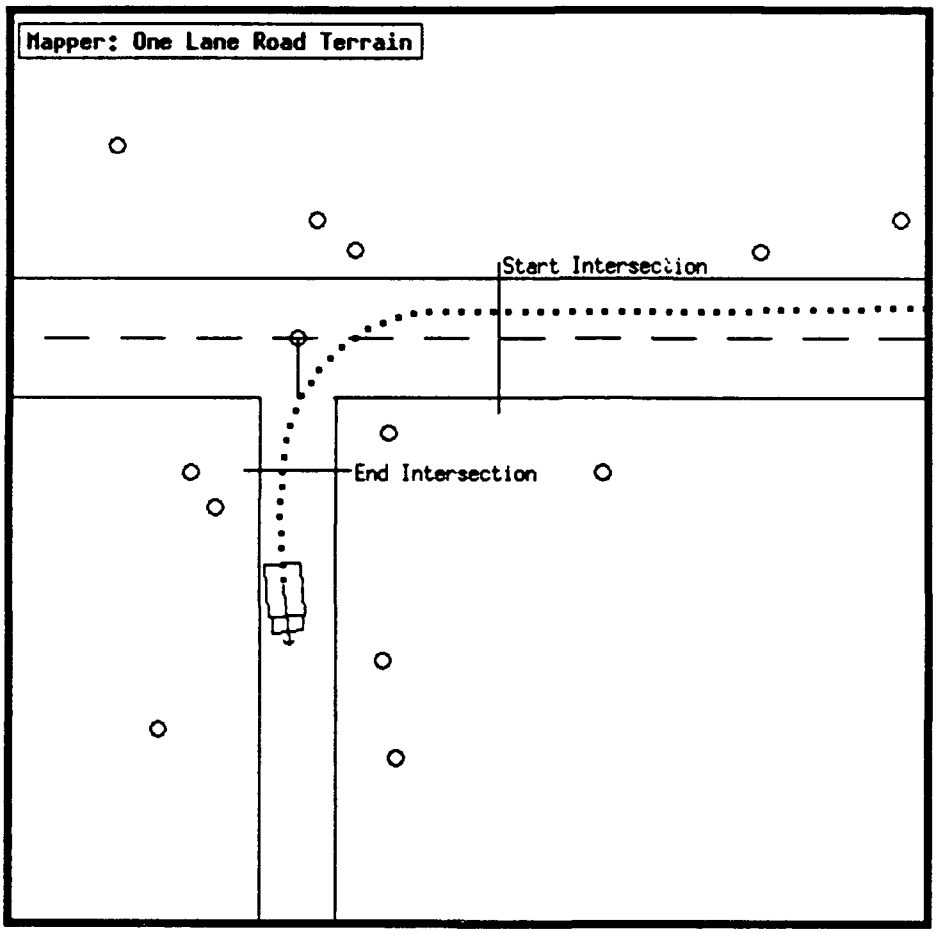


Figure 7.2: A section of a map created and maintained by the annotated map system. The map shows the vehicle traversing an intersection between a single- and a two-lane road. The lines across the roads are alarms which are triggered when crossed by the vehicle. Triggering an alarm results in a message being passed from the map manager to the arbitrator indicating a change in terrain type. The circles on the map represent the positions of landmarks, such as trees and mailboxes. The annotated map system uses the locations of known landmarks to correct for vehicle positioning errors which accumulate over time.

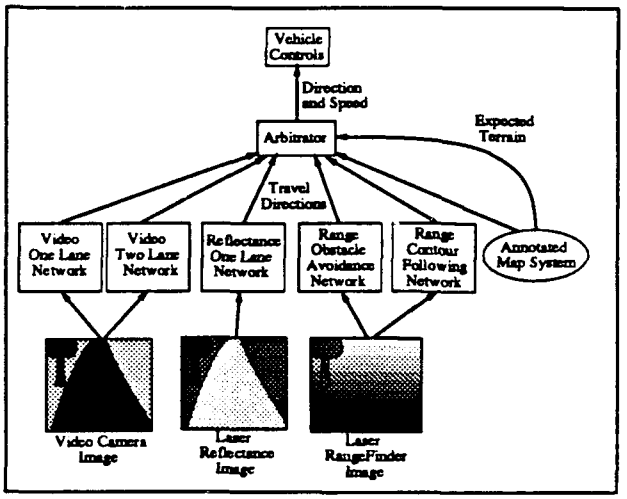


Figure 7.3: The integrated ALVINN architecture. The arbitrator uses the terrain information provided by the annotated map system as well as symbolic models of the driving networks' capabilities and priorities to determine the appropriate module for controlling the vehicle in the current situation.

and start listening to the two-lane road following network. When the alarm is triggered during the run, the arbitrator combines the advice from the annotated map system with the steering directions of the neural network modules using a technique called relevancy arbitration.

Relevancy arbitration is a straightforward idea. If the annotated map system indicates the vehicle is on a two-lane road, the arbitrator will steer in the direction dictated by the two-lane road driving network, since it is the relevant module for the current situation. If the annotated map system indicates the vehicle is approaching an intersection, the arbitrator will choose to steer in the direction dictated by the annotated map system, since it is the module that knows which way to go in order to head towards the destination. In short, the arbitrator combines symbolic knowledge of driving module capabilities with knowledge of the present terrain to determine the relevant module for the current circumstances.

The relevancy of a module need not be based solely on the current terrain information provided by the annotated map system. Instead, the arbitrator also employs rules for determining a module's relevancy from the content of the module's output. The obstacle avoidance network has one such rule associated with it.

The obstacle avoidance network is trained to steer straight when the terrain ahead is clear and to swerve to prevent collisions when confronted with obstacles. The arbitrator gives low relevancy to the obstacle avoidance network when it suggests a straight steering direction, since the arbitrator realizes it is not an applicable knowledge source in this situation. But when it suggests a sharp turn, indicating there is an obstacle in the vehicle's path, the urgency of avoiding a collision takes precedence over other possible actions, and the steering direction is determined by the obstacle avoidance network. This priority arbitration is similar in many ways to the subsumption architecture [Brooks, 1986], although the most common interaction between behaviors in Brooks' systems is for higher level behaviors to override less sophisticated, instinctual ones.

By combining map-related knowledge about the current driving situation with knowledge about abilities and priorities of individual driving modules, the integrated architecture provides the system with capabilities that far exceed those of individual driving modules alone. Using this architecture, the system has successfully followed a 1/2 mile path through a suburban neighborhood from one specific house to another. In navigating the route, the system was required to drive through three intersections onto three different roads while swerving to avoid parked cars along the way. At the end, the vehicle came to rest just one meter from its intended destination.

7.3 Analysis and Discussion

Rule-based integration of multiple expert networks has significant advantages connectionist arbitration schemes developed by other. One such advantage is the ease of adding new modules to the system. Using rule-based arbitration, the new module can be trained in isolation to become an expert in a new domain, and then integrated by writing rules for the arbitrator which specify the new module's area of expertise and its priority. This is in contrast to other connectionist expert integration techniques, such as the task decomposition architecture [Jacobs et al., 1990, Jacobs et al., 1991], connectionist glue [Waibel, 1989] and the meta-pi architecture [Hampshire & Waibel, 1992]. To combine experts using these techniques requires the training of additional neural network structures, either simultaneously with the training of the experts in the case of the task decomposition architecture, or after expert training in the case of the connectionist glue and meta-pi architectures. Adding a new expert using these techniques requires

retraining the entire integrating structure from scratch, which involves presenting the system patterns from each of the experts' domains, not just the new one. This large scale retraining is particularly difficult in a task like autonomous navigation because it requires either driving over all the experts' domains again, or storing a large number of domain-specific images for later reuse.

Another significant advantage of rule-based arbitration is the ease with which non-neural network knowledge sources can be integrated into the system. Symbolic tasks such as planning and reasoning about a map are currently difficult to implement using neural networks. In the future, it should be possible to implement more symbolic processing using connectionist techniques, but until then, rule-based arbitration provides a way of bridging the gap between neural networks and traditional AI systems.

Rule based multi-network arbitration is not without shortcomings however. The current implementation relies too heavily on the accuracy of the annotated map system, particularly for negotiating intersections. The question might be asked, why is the mapping system required for intersection traversal in the first place? Why can't the driving networks handle intersections? When approaching an intersection, an individual driving network will often provide ambiguous steering commands, since there are multiple possible roads to follow. If left on its own, a road-following network will often alternately steer towards one or the other road choices, causing the vehicle to oscillate and eventually drive off the road. In addition, even if the network could learn to definitively choose one of the branches to follow, it still wouldn't know which is the *appropriate* branch to choose in order to head toward the destination. In short, the mapping modules can be viewed both as a useful source of high level symbolic knowledge, and as an interim solution to the difficult perceptual task of intersection navigation.

The annotated map system as currently implemented is not a perfect solution to the problem of high level guidance because it requires both detailed knowledge of the route and an accurate idea of the current vehicle position. In certain controlled circumstances, such as rural mail delivery, the same route is followed repeatedly, making an accurate map of the domain feasible. However a system capable of following less precise directions, like "go about a half mile and turn left on Seneca Road", is clearly desirable. Such a system would require more reliance on observations from perception modules and less reliance on knowledge of the vehicle's exact position when making high level decisions.

Conceptually, this shift towards reliance on perception for high level guidance could be done with or without a map. Observations of objects like the Seneca

Road street sign could be used to update the vehicle's estimated location. In fact, position updates based on perceptual observations are currently employed by the annotated map system when it triangulates the vehicle's location based on the positions of known landmarks in laser range images. But position updates are only helpful when the observations are location specific. When objects such as stop lights, or arbitrarily located objects like "road construction ahead" signs appear, the system's response should be independent of the vehicle's location.

These location-independent observations could be modeled as positionless alarms in the annotated map. When a perception module sees an object like a "road construction ahead" sign, it would notify the map manager. The map manager would treat the sighting as an alarm, distributing the information associated with the alarm to the pertinent modules. Perception triggered alarms would allow the system to transition between its current perceptual abilities and future, more advanced capabilities.

Although ALVINN is not yet capable of identifying and reading individual signs in order to pinpoint the vehicle's location, in the next two chapters I describe connectionist techniques which allow perceptual observations by the networks to help guide high level reasoning. These techniques, described in the next two chapters, allow the networks to estimate their own reliability in the current situation. These reliability estimates can be used to refine the vehicle's current position estimate. When coupled with the symbolic integration techniques described in this chapter, they greatly extend the flexibility and reliability of the ALVINN system.

The connectionist arbitration techniques in the next two chapters also solve another shortcoming of rule-based arbitration, the binary nature of its integration procedure. Currently, a module is deemed by the annotated map system as either appropriate or inappropriate for the current road type. This binary decision does not address the question of intelligently combining modules trained for the same domain, such as the video-based single-lane driving network and the laser reflectance-based single-lane driving network. There are obviously some situations, such as night driving, when one network is better suited than the other. One possible way to take more subtle circumstances into account when weighting the steering directions from multiple networks, would be to augment the arbitration rules to consider more context than just the current road type. However this approach would increase ALVINN's dependence on difficult to acquire symbolic knowledge. The connectionist reliability estimation techniques described in the next two chapters provide a more natural method of combining the outputs from multiple networks.

Chapter 8

Output Appearance Reliability Estimation

In the last chapter, rule-based arbitration was shown to be a useful technique for integrating multiple driving networks. However the technique was also shown to have significant limitations, including over-reliance on detailed map knowledge and an inability to smoothly combine the outputs from multiple networks. In this chapter I present a connectionist arbitration technique, called *Output Appearance Reliability Estimation* (OARE), which avoids these shortcomings.

The OARE technique estimates the reliability of individual networks and then uses these estimates to weight the outputs from multiple networks. Because it provides an estimate of an individual network's absolute reliability, and not just an indication of the relative reliabilities of multiple networks, it is useful for more than just arbitration. In the domain of autonomous navigation, OARE can also be used to:

- Refine the vehicle's estimated position
- Control the vehicle's speed
- Determine when a new network needs to be trained

In the first section, I review previous connectionist techniques for multi-network arbitration, pointing out their shortcomings with regard to autonomous navigation. Next I present the details of the OARE technique, along with results obtained using it in ALVINN. I then discuss the conditions necessary for OARE to

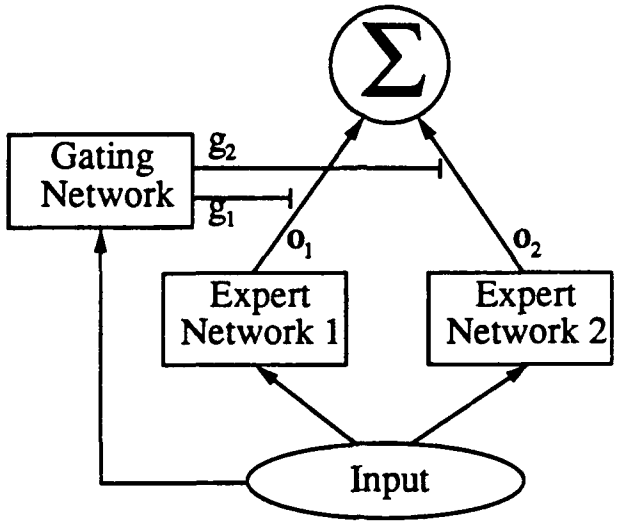


Figure 8.1: The gating network architecture.

be an effective method for multi-network integration. Finally I present shortcomings of OARE as a motivation for another network reliability estimation technique, to be discussed in the following chapter.

8.1 Review of Previous Arbitration Techniques

Previous connectionist arbitration techniques have been based primarily on gating networks [Jacobs et al., 1991, Hampshire & Waibel, 1992]. The underlying idea is illustrated in Figure 8.1. In this architecture, the gating network receives the same input as the expert networks. But instead of contributing directly to the final output of the system, it provides gating signals, labeled g_1 and g_2 in Figure 8.1, which are used to weight the outputs from the respective expert networks to determine the final response using the formula:

$$\mathbf{o} = \sum_i g_i \mathbf{o}_i$$

where $\mathbf{o}$ is the final output vector, g_i is the gating signal for expert i and $\mathbf{o}_i$ is the output vector of expert i.

Jacobs et al. [Jacobs et al., 1991] employ the architecture shown in Figure 8.1 in their competitive expert mixing models. In these models, the gating network and the expert networks are trained simultaneously using back-propagation with a modified error term to encourage competition among the experts. During learning, the experts compete to respond to individual input patterns. The contribution of an expert i that does better than average at predicting the desired output is increased by raising its gating signal g_i. At the same time, each expert is trained to better predict the desired output for those patterns in which its contribution to the final output is large. In other words, the error signal back-propagated through an expert network i for a particular pattern is scaled by the gating signal for that expert, g_i. Deriving the gating function through competitive learning encourages the experts to specialize for coherent subtasks, a useful property in complex domains where the appropriate task decomposition is not known a priori.

In Hampshire and Waibel's Meta-Pi architecture, experts are trained to co-operate rather than compete. The individual expert networks are first trained separately on a pre-specified subset of training patterns, such as the speech input from a single speaker. Each expert network produces what it considers to be the correct response for the given input, labeled $\{\rho_1, \rho_2, \ldots, \rho_k\}$ in Figure 8.2. The outputs from the individual networks are then combined via multiplicative connections from the gating network, labeled $\{M_{\pi_1}, M_{\pi_2}, \ldots, M_{\pi_k}\}$ in Figure 8.2, to determine the final response. A learning rule similar to the Sigma-Pi extension to back-propagation [Rumelhart, Hinton & Williams, 1986] is used to train the multiplicative connections and the internal units of the gating network. They are trained to minimize the error in the final response across patterns drawn from all the experts' domains of knowledge. Using this technique, Hampshire has shown that the Meta-Pi gating architecture can learn to recognize speech from a male speaker using a combination of speaker-specific modules trained on *other* speakers.

While interesting and useful, both the competitive expert mixing techniques and the Meta-Pi architecture suffer from significant shortcomings, particularly for the domain of autonomous navigation. First, they require substantial additional network structure and training. The Meta-Pi strategy involves a computationally expensive training phase during which the gating network is presented with exemplars from all the subdomains of the task. This is impractical for on-line training in autonomous navigation, since it would require storage of an extensive corpus of patterns from previously encountered driving domains, and tedious training of the gating network after training the expert networks.

In the competitive expert mixing model, the experts and gating network can

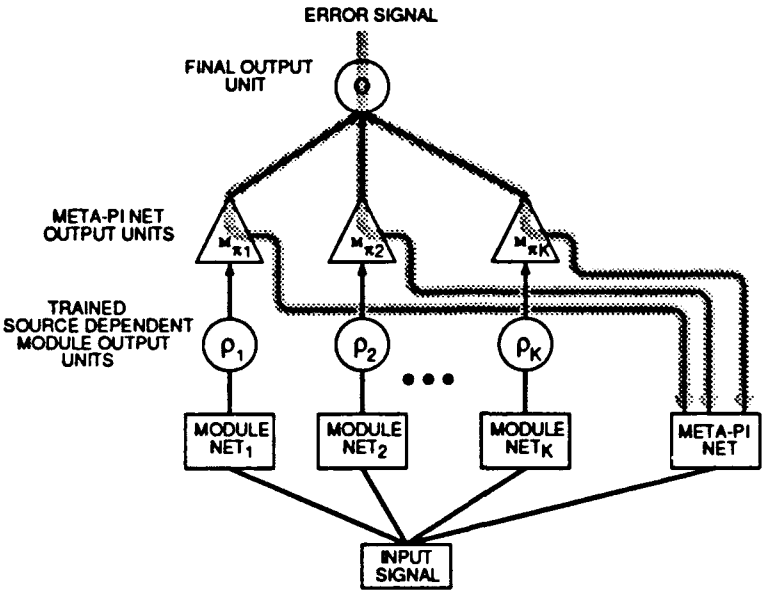


Figure 8.2: The Meta-Pi Architecture [Hampshire & Waibel, 1992].

be trained simultaneously, but at a high price. Simultaneous training is very expensive, since both the forward and backward passes of back-propagation must be performed for each pattern for each expert network. In a sense, simultaneous training defeats an important purpose of a modular architecture, namely the ability to efficiently and independently train sub-modules. While it is important for the system to automatically allocate new networks when the existing ones are unable to handle the current situation, it is impractical to expect all the networks that might eventually be required by the system to be implemented and simulated throughout the training process.

In the experiments performed by Jacobs et al. using the competitive expert mixing model, the architecture was trained from the very start on examples covering the full range of tasks. It is not clear that the competitive expert mixing model can "hold networks in reserve" until a new situations are encountered which require them. In real world applications like autonomous navigation, where unexpected situations frequently arise, all the experts and the gating network may require retraining from scratch if the initial task decomposition is insufficient.

The Meta-Pi architecture suffers from a similar shortcoming. In this case,

adding a new expert by definition requires restructuring and retraining the gating network. The expense of adding new experts in the Meta-Pi architecture may not be much of a problem when a sufficiently large ensemble of experts is collected to allow accurate processing of novel situations by a combination of previously trained experts. However, this synergy between multiple experts when presented with new stimuli has yet to be demonstrated on a large-scale problem.

Finally, both the Meta-Pi architecture and the competitive expert mixing model provide only an indication of a network's applicability for the current input *relative to other networks*. They do not provide an absolute measure an expert's relevance. Therefore, these techniques cannot easily indicate when none of the ensemble of experts is appropriate and therefore a new expert is required. On the contrary, both architectures will happily combine the outputs from their experts to produce what they consider to be the appropriate response, but which may very well be incorrect.

The ability to accurately predict impending failures is critical in domains like autonomous driving where mistakes can be catastrophic. Ideally, the individual experts should produce both a response to an input pattern and an estimate of the response's probability of correctness. This reliability estimate could be used both to weight the output from multiple experts and to alert a human observer when a new expert needs to be trained. The technique would be most useful if it required little or no additional network structure or training. In the remainder of this chapter I describe a technique called *Output Appearance Reliability Estimation* (OARE) that meets these criteria and results obtained applying it to autonomous navigation.

8.2 OARE Details

OARE employs the simple idea that if the network's output is misshapen, the network is probably confused. Specifically, after presenting the network with an input pattern, the network's output is compared with the nearest ideal Gaussian output pattern. The farther the actual output is from the ideal output, the lower the estimated reliability of the network's response. While very straightforward, this technique can quite accurately estimate the reliability of multi-layered perceptrons under certain conditions described below.

The first step in estimating a network's reliability using the appearance of its output is to determine the nearest ideal output pattern. For 1-of-N classification tasks like speech or character recognition, the nearest ideal output pattern is easy to

determine. It is the pattern in which the output unit with the maximum activation is fully activated (i.e., 1.0 activation level) and the remaining units are fully inhibited (i.e., -1.0 activation level). For more complex output representations, like the gaussian output representation employed by ALVINN, determining the nearest ideal output is slightly more difficult. It involves finding the output pattern with the gaussian activity distribution described in Chapter 2 which best fits the network's actual output pattern. The same least squares matching technique described in Chapter 2 for interpolating a network's dictated steering direction can be employed in OARE to determine the best fit gaussian output vector.

Once the closest ideal output vector is determined, various metrics such as Hamming distance, Euclidean distance or sum-squared difference can be used to measure the closeness of the actual output vector to the nearest ideal one. I have rather arbitrarily chosen to employ sum-squared difference as the distance metric in OARE. Mathematically, if net_i^a is the actual net input to and output unit i before application of the sigmoid squashing function and net_i^d is its desired net input, then the network's output appearance error E_a is expressed as

$$E_a = \sum_i (net_i^a - net_i^d)^2$$

Using the actual and desired net input to the output units before the sigmoid squashing function when calculating the appearance error significantly increases the correlation between the output appearance error and the network's probability of responding incorrectly. This improvement stems from the fact that the sigmoid's non-linearity distorts the similarity metric between two output patterns. Consider a single unit in the following two scenarios. In the first, the unit's activation level is 0.95 and its desired level is 0.99. In the second case the unit's activation level is 0.50 and the its desired level is 0.54. Despite the fact that the change in activation level is the same in both cases, the first would require a much larger change in the internal representation (given fixed weights) to produce the desired change in response. As a result, the hidden representation in the first case should be considered much less appropriate than in the second, since it would require a larger change in the first to correct the error than in the second.

Put another way, the first indication of a network's confusion is the disappearance of the peak at the center of the gaussian. In effect, the gaussian gets chopped off. A single direction still receives support, and it is sufficient support to create a "hill" of activation in the output vector, but the support is not strong enough to drive the center units in the gaussian to extreme values. When employing activation

values to compute output appearance error, the disappearance of the gaussian's peak does not lead to a large increase in appearance error, since the differences in activation values are not very large between the ideal output gaussian and the actual, chopped-off gaussian. In a sense, this subtle sign of possible error gets washed out. Only on very confusing images, like at intersections, is support for a single steering direction low enough to make the activation levels of the output units different substantially from the ideal values. On these images, the appearance error rises rapidly as network confusion increases, since a small decrease in support for a steering direction when in the sigmoid's linear range results in a large decrease in activation level and hence a large increase in appearance error. Using the net input to a unit instead of the unit's activation level when calculating output appearance error prevents the sigmoid's non-linearity from distorting the measure of support for its steering direction. In terms of performance, using net input instead of activation level in OARE results in an increase of 0.1 to 0.2 in the correlation coefficient between a network's output appearance error and its likelihood of responding incorrectly (see the next section for more details).

8.3 Results Using OARE

Empirically I have found a driving network's output appearance error to be a good indication of its reliability. Figure 8.3 illustrates the strong correlation between the output appearance error and the magnitude of the steering errors a trained network makes when driving over a 300 meter stretch of road. The graph was made by having a person drive while recording both the person's steering direction and the steering direction dictated by the network. The dashed line represents the network's output appearance error at various points along the road. The solid line represents the error in the steering direction dictated by the network, as measured by the difference in turn curvature between the steering directions of the network and the person. Specifically,

$$E_s = \left| \frac{1}{r_p} - \frac{1}{r_n} \right|$$

where E_s is the network's steering error, r_p is the person's steering radius, and r_n is the network's dictated steering radius. As can be seen from the graph, the degree of degradation in the shape of the output vector is closely related to network steering error. In fact, over a number of trials on a number of different road types,

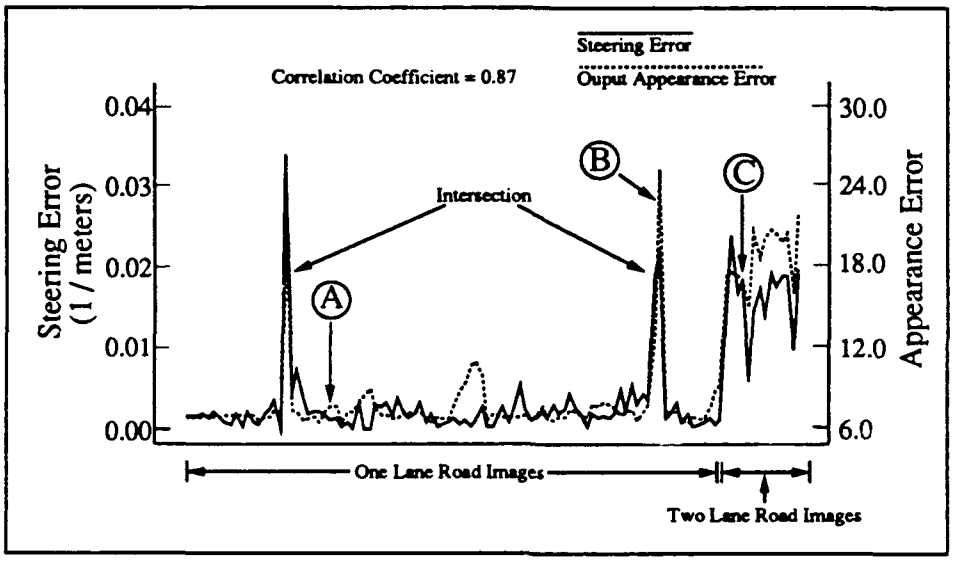


Figure 8.3: Graph comparing steering error (solid line) and output appearance error (dotted line) over a 500 meter stretch of road. As can be seen from the high correlation between the two curves, when the appearance of the output is far from ideal, the network is likely to be making a significant steering error.

I have found the average correlation coefficient between output appearance error and steering error is 0.82.

The reliability prediction provided by output appearance can be employed to improve the performance of a neural network-based autonomous driving system in a number of ways. The simplest use of OARE is to control vehicle speed. The more ideal the output, the more confident the network, and hence the faster the vehicle is allowed to travel under autonomous control.

A second use for a reliability estimate is to update the vehicle's position on a map. When the vehicle encounters a confusing situation such as an intersection, the network's output appearance error rises dramatically. This effect is illustrated by the two sharp peaks labeled with "intersection" in Figure 8.3. At these two points, the vehicle encounters a fork in the road and the network becomes confused. The situation and response from the network at the second intersection, labeled "B" in Figure 8.3, is shown in the middle frame of Figure 8.4. Since there are two possible directions in which to steer, the network produces an output vector with a bimodal distribution. This ambiguous output results in a large output appearance

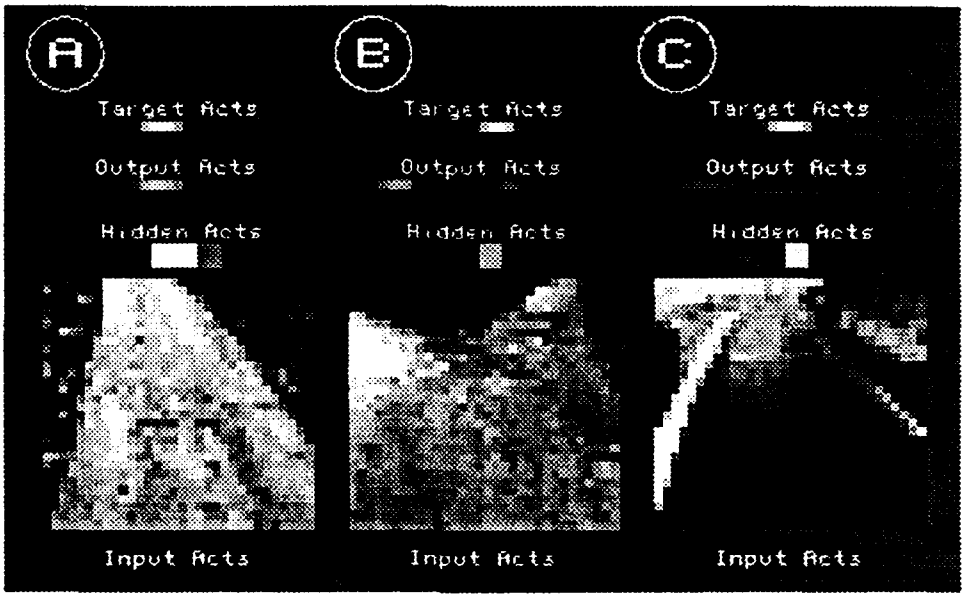


Figure 8.4: The situation and ALVINN's response at the three points labeled A, B and C in the previous Figure. Notice the bimodal distribution of the network's output vector for the middle, fork image. Also, notice the absence of a sharp peak in the output vector for the two-lane road image on the right.

error since it does not resemble a single gaussian peak. The sudden increase in output appearance error is a good indication that the vehicle has just reached a confusing location. By watching for this spike in output appearance error, the annotated map module described in Chapter 7 is able to accurately pinpoint the vehicle's location *without a highly detailed or accurate map*.

Recall from Chapter 7 that the situation that has most required a highly accurate map is intersection traversal. The previous method for negotiating intersections relied solely on the mapping module to steer the vehicle correctly, since the networks were not reliable at intersections. More reliable intersection traversal can be achieved by combining the annotated map module's high-level route knowledge with the road following information contained in the ambiguous output vector.

When the vehicle reaches an intersection, the mapping module's knowledge that the vehicle should turn right in order to head towards the goal can bias the arbitrator towards choosing the peak on the right of the network's output vector as the direction to steer. Using this technique, the architecture shown in Figure 8.5

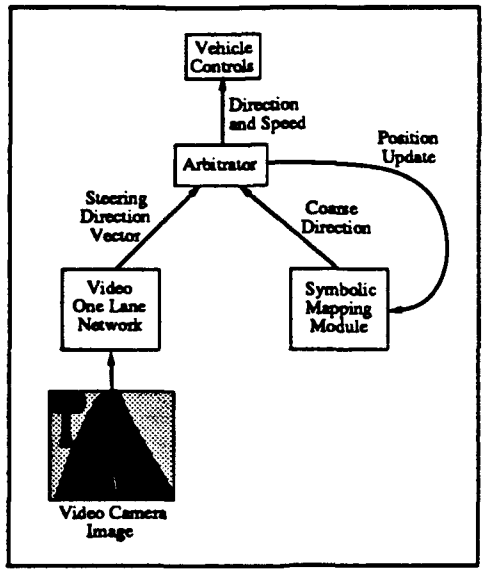


Figure 8.5: Architecture used with OARE to navigate over a pre-planned route through a network of single lane roads.

has successfully driven over a 400 meter single lane course containing three intersections. The arbitrator realized when it had reached each intersection by observing a sudden jump in the network's output appearance error, at which point it sent a message to the mapping system telling it the intersection had been reached. It then selected from among the ambiguous steering directions provided by the network on the basis of coarse commands from the symbolic mapping module like "turn right at this intersection". Using output appearance reliability estimation to more closely couple the symbolic knowledge of the mapping module with the steering expertise of an artificial neural network greatly extends the flexibility and robustness of a single network autonomous navigation system.

But perhaps the most useful application of OARE is in arbitrating between multiple expert networks. When a network trained to drive in one situation is presented with images from another situation, the appearance of the output is far from ideal. This is illustrated by the high output appearance error for the two-lane road images on the right side of the graph in Figure 8.3. The single lane network's response on a typical two-lane road image, labeled as point C in Figure 8.3, is shown in the right frame of Figure 8.4. In this case, the network's

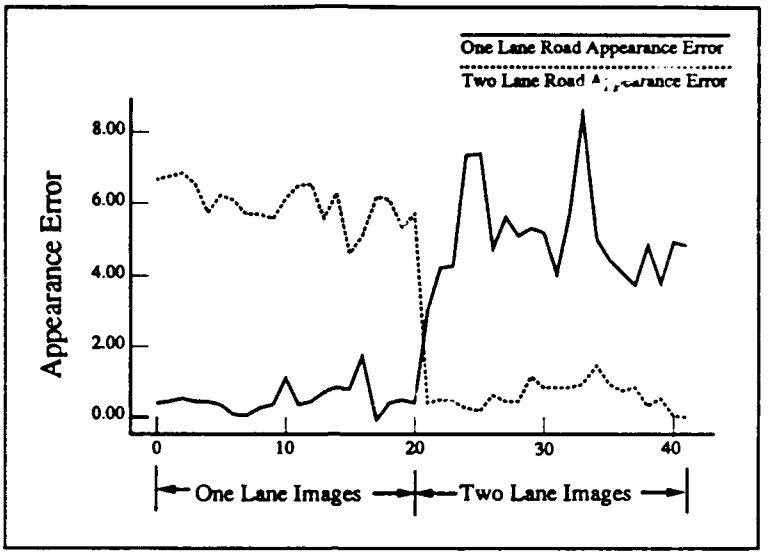


Figure 8.6: Graph comparing the appearance error of two networks trained for different situations. The network trained for single lane roads produces an output vector that appears nearly ideal on single lane road images (the first 20) and an ambiguous output vector on two-lane road images (the last 20). The opposite is true for the network trained for two-lane road images.

output supports a single, left steering direction. However the sharp peak in the output vector, characteristic of a confident network, is missing. This missing peak can be detected by the output appearance calculation described above and used to discount the response of this network.

Figure 8.6 demonstrates in more detail the domain-specific nature of networks' output appearance errors. The first 20 images are from a stretch of single lane road, while the second 20 are from a stretch of two-lane road. Notice the network trained for a single lane road provides nearly ideal output responses on the single lane road images, but ambiguous responses on the two-lane road images. The opposite is true for the network trained on two-lane roads.

I have utilized this output appearance domain-dependency as an arbitration mechanism for a multi-network ALVINN system which included a single-lane video based driving network, a single lane laser reflectance-based driving network, and a two-lane video-based driving network. The arbitrator was able to use the output appearance error of the networks to select the most appropriate one for the

current situation. A beneficial side effect of employing OARE for arbitration is fault tolerance with respect to sensor failure. While driving autonomously on a single-lane road, ALVINN is able to smoothly switch from relying on the single-lane video network to the laser reflectance network when the signal from the video camera is lost.

In summary, Output Appearance Reliability Estimation is a computationally inexpensive way to determine the likelihood and magnitude of errors made by driving networks in the ALVINN system. OARE improves the system's flexibility and robustness by providing a means for

- Intelligently controlling vehicle speed
- Pinpointing vehicle location
- Arbitrating between multiple networks

8.3.1 When and Why OARE Works

A technique similar to OARE had been tried with little success on multi-class pattern recognition tasks such as character recognition [Linden & Kindermann, 1989] and speech recognition [Hampshire, 1990]. Linden and Kindermann found that networks trained to perform digit recognition would often unambiguously classify random input patterns as particular digits, so the appearance of the output response was a poor indication of the network's correctness. In contrast, I have found that output appearance is a very good indication of the reliability of ALVINN driving networks.

One possible explanation for the difficulties others have experienced with OARE is that they employed the *output activations*, after applying the sigmoid squashing function, to quantify output appearance. Recall that when using output activations for OARE in the autonomous navigation domain, similar poor results were obtained. Only after employing the net input to the output units, instead of their activations, did output appearance error correlate highly with network reliability. But this is probably not the entire explanation. Additional reasons for the failure of OARE in character and speech recognition stem from both characteristics of the tasks themselves and the connectionist architectures employed to solve them.

Two additional conditions necessary for OARE to be effective are that 1) coherent support from multiple input features for a particular response be a sufficient

condition for correct classification and 2) an ideal appearing output vector should only result from coherent support from multiple features for a single response. If either of these conditions is violated, as will be demonstrated for classification tasks like character and speech recognition, then output appearance will not be a good indication of network reliability.

Task Requirements

Consider the task of handwritten digit recognition. In this domain, discriminations between one class and another (say between a 5 and a 6) must often be based on the activations of only a few input units. For a connectionist classifier to work on this task, those few crucial units must be capable of greatly influencing the output of the network, since they must change its output classification. With a small combination of units able to greatly influence the network's output, it is easy to see how a random or unfamiliar input pattern, which happens to have the right combination of activations for these few crucial units, could be unambiguously classified as a particular digit. This is in fact just Linden and Kindermann found. A few important input units could determine whether a pattern would be classified as a particular digit, regardless of the activation of the other units.

In short, the highly non-linear relationship between input feature combinations and output classification in the digit recognition task violates both necessary conditions for OARE to be effective. Coherent support from multiple features is not sufficient for correct classification, since when the digit is a "6", the class "5" will likely receive nearly as much support from input features as the "6" class. Also, the very subtle discriminations necessary for correct classification requires that subtle individual features in the input be capable of greatly influencing the output vector. Therefore, just one or two could features in the input could produce an ideal appearing output vector, violating the second condition for OARE's effectiveness.

In autonomous navigation, there is a much more direct relationship between input features and desired output, so OARE works well. The crucial difference between this domain and digit recognition is that the correct output is determined not by small localized input features, but by the combination of many consistent features such as the positions of the lines painted on the road and the locations of the boundaries between the road and the non-road. No single feature is necessary for determining the correct output. Instead, many features with small individual importance contribute to determining the correct steering direction. Since no

single feature is important enough (i.e., has large enough associated weights) to determine the correct output in isolation, the only way for the network to produce an ideal appearing output is for many features to support the same steering direction. At the same time, if many features support the same output, that output is likely to be correct, making an ideal appearing output vector a strong indication that the output is correct. Conversely, an ambiguous output vector will only result when no single steering direction receives coherent support from multiple input features, meaning the network is confused. Again, the appearance of the output correlates well with the likelihood of the output's correctness.

Output Representation Requirements

Task characteristics alone are not sufficient to guarantee the effectiveness of output appearance reliability estimation. The output representation chosen for the task is also crucial. If the output representation does not allow for comparison with an ideal output, or does not encourage the network to give small weight to individual input features, than OARE can still fail. Such is the case with the two alternative output representations explored in Chapter 2.

Recall in the so called "graded single unit" output representation the steering direction is encoded in the activation level of a single output unit. Obviously in this representation there can be no notion of an ideal output vector, since each activation level represents a different entirely legitimate steering direction.

The problem with the "one-of-N" output representation is that it ignores the output's structured nature by treating each output unit as if it represented a discrete class unrelated to its neighbors. As was seen in Chapter 2, this division of steering directions into disjoint classes converts the problem into a highly non-linear classification task. Just as in the digit recognition problem, unambiguous classification of a road image into one of N distinct classes requires the network to make discriminations based on very subtle input features. This heavy weighting of small features can result in unambiguous classification without coherent support from multiple features, violating the second necessary condition for OARE's effectiveness. As a result, the correlation between output appearance error and steering error when using a one-of-N representation averages only about 0.3, less than half that achieved with the gaussian output representation (0.82).

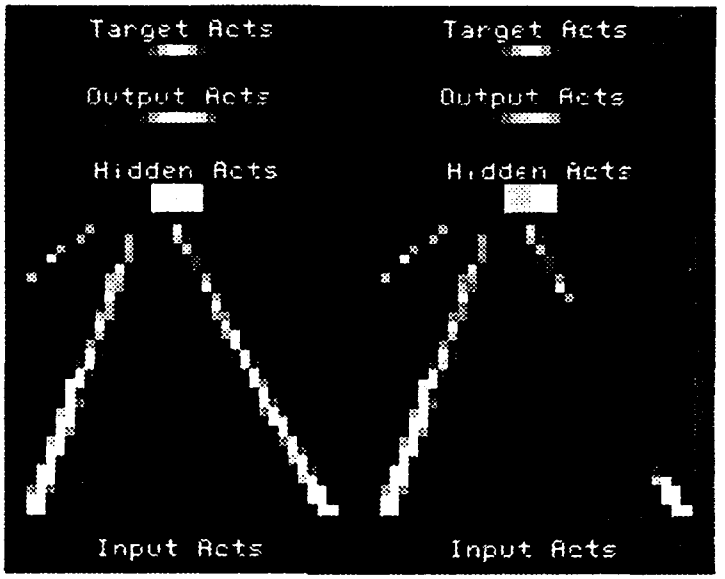


Figure 8.7: The response of a network trained on artificial two-lane road images in two situations, a "normal" image on the left, and a novel image on the right. The network's output vector is not significantly different between the two images.

8.4 Shortcomings of OARE

OARE has proven quite useful in the domain of autonomous navigation, but it still has some significant shortcomings. While the correlation between output appearance error and steering error is high, there are still situations where the network's output appears reliable but the input image is far from typical. Such a scenario is illustrated in Figure 8.7. The network was trained on artificial two-lane lined road images like the one shown on the left. When presented with an image like the one on the right, with a large section of the right-hand lane marker missing, the network's output response was nearly indistinguishable from its response on the "normal" image. In fact, on this particular image, the network's response was slightly more "ideal" on the right image than the left.

In one sense, the network's confident response to the image on the right is forgivable, since it has chosen the only plausible generalization from the training data for this novel circumstance: steer straight ahead. But as was seen above, we'd like to glean more from the network than just the correct direction to steer.

We'd like an indication of the situation's novelty, for use by high-level reasoning systems like the annotated map module. In this specific situation, a gap in the right lane marker might signal an off-ramp which the vehicle should follow. Without an indication of the situation's novelty, the annotated map module would either have to rely on accurate map and vehicle position information, or miss the exit entirely.

Even when the network indicates uncertainty through a less than ideal output vector, the confusing aspects of the input are often impossible to determine because of the difficulty in correlating output activation profiles with specific spurious input conditions. The bimodal output distribution at intersections is an exception. It is usually very difficult to determine what features in the input are confusing the network. In the situation depicted in the left image of Figure 8.7, if the network *did* produce an output vector with the gaussian peak chopped off, it would be impossible to determine if its confusion was caused by a passing car occluding the centerline, or a gap in the right boundary signaling an exit ramp.

In the next chapter, I present a technique called Input Reconstruction Reliability Estimation (IRRE) which alleviates these problems by determining both the degree of novelty of the current input and the identity of the novel features.

Chapter 9

Input Reconstruction Reliability Estimation

The ability to identify and reason about novel aspects of their input would greatly enhance the capabilities of artificial neural networks. The extent of the novelty could be used to judge the appropriateness of individual networks for the task to be performed. The location and shape of novel features could be employed to identify the unusual components of the input and to choose an appropriate response.

This chapter describes a technique called Input Reconstruction Reliability Estimation (IRRE) which improves on the Output Appearance Reliability Estimation technique described in the last chapter. Like OARE, the IRRE technique can accurately estimate the reliability of a network's response. But in addition, IRRE provides useful details about the reason for the network's confusion in a novel situation.

9.1 The IRRE Idea

The hidden representation in artificial neural networks is a compressed representation of important input features. In the case of ALVINN networks, the hidden units represent the position and orientation of important features like the road edges and lane markers. If this internal representation of what ALVINN "thinks it is seeing" could be compared with the actual input, the amount of discrepancies would be a good indication of the situation's novelty. Put another way, the severely limited number of units in the hidden layer prevents the network from accurately

representing an arbitrary input pattern. Instead, the hidden units learn to devote their limited representational capabilities to encoding the position and orientation of consistent, frequently-occurring features from the training set. When presented with an atypical input, the feature detectors developed by the hidden units will not accurately cover or account for all the actual input features.

This type of situation is depicted in Figure 9.1. It shows the weights of a network trained on artificial two-lane roads like the ones in Figure 8.7. Notice that the feature detectors developed by the hidden units all represent *unbroken* lane markers. This is because all the training patterns showed roads with continuous lines. The technique described below uses this type of constraint on the appearance of typical inputs provided by the hidden representation to reconstruct the nearest "ideal" exemplar to the actual input. This reconstruction is shown in the block labeled "Reconstructed Input" in Figure 9.1. Like the hidden unit receptive fields and unlike the actual input, the reconstructed input contains an unbroken right lane marker. The degree and location of discrepancies between reconstructed and actual inputs can be used to measure and reason about the novelty of the current situation.

The question is how to reconstruct the nearest familiar input pattern from the network's internal representation. The next three sections describe technique I have experimented with for performing this reconstruction.

9.2 Network Inversion

The first reconstruction technique uses is similar to the "network inversion" method developed by Linden and Kindermann [Linden & Kindermann, 1989]. In this method the reconstructed input vector is computed iteratively by back-propagating an error signal from the outputs all the way to the input units.

In more detail, the first step in the modified network inversion procedure is to present the network with the current input pattern and propagate activation to the output units. The nearest ideal output vector is then determined using the techniques described for OARE. The difference between the actual output vector and the nearest ideal output vector is used as an error signal in the back-propagation phase of network inversion. This error signal is propagated all the way down to the input layer using the delta equations from the back-propagation gradient descent algorithm. For an output unit j,

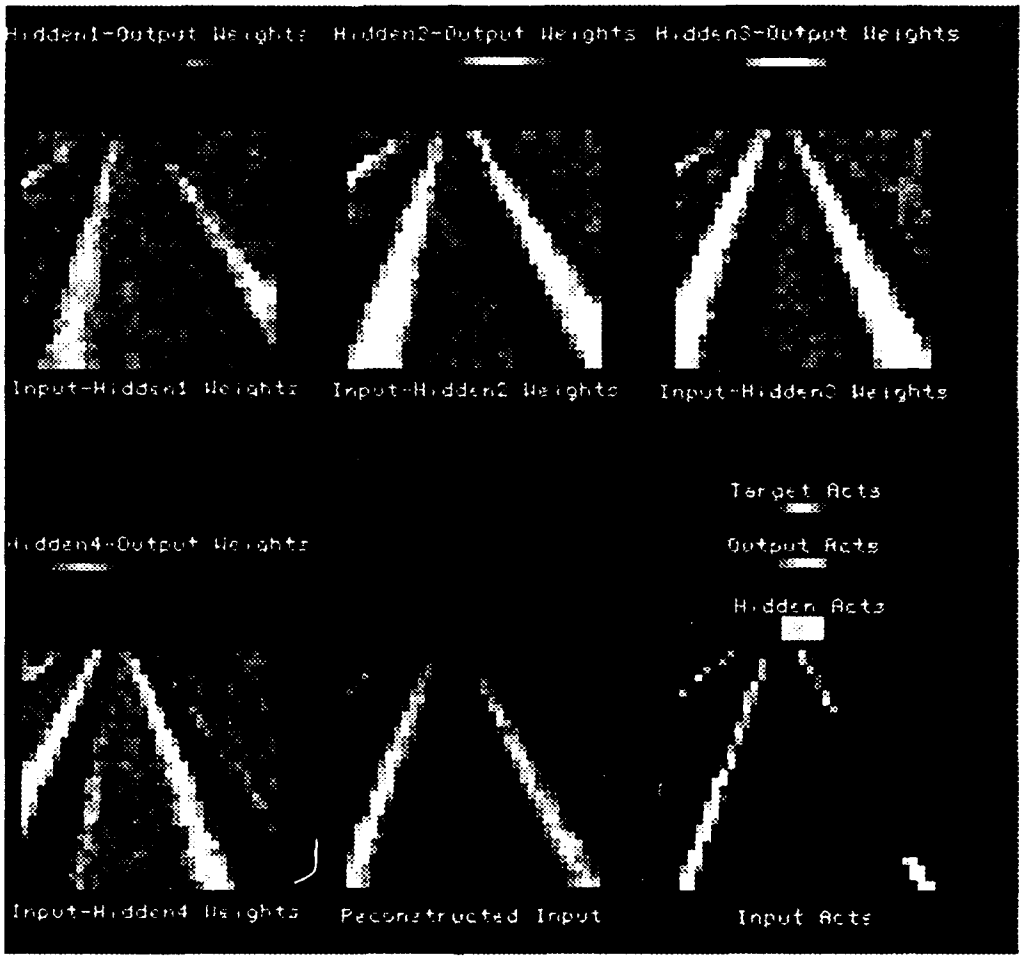


Figure 9.1: A network trained on artificial two-lane roads with continuous lane markers. When presented with an image containing a gap in the right lane marker, the image produced by the input reconstruction process "fills in" the gap to produce the familiar input pattern nearest to the actual input.

$$\delta_j = (t_j - o_j) f'(net_j)$$

where δ_j is the error signal for output unit j, t_j is the "ideal" activation level for unit j, o_j is the actual activation for unit j, and $f'(net_j)$ is the derivative of the activation function.

If unit j is a hidden or input unit,

$$\delta_j = f'(net_j) \sum_k \delta_k w_{jk}$$

where k represents a unit in the layer above unit j, and w_{jk} is the weight of the connection from unit j to unit k. The δ values for the input units are used to determine the magnitude and direction of change to their activations in order to decrease the difference between the actual and ideal output response. In order to keep the reconstructed activation levels of the input units bounded within their normal range of -1 to 1, the activation levels of the input units are not directly altered using their error signals. Instead, a net input component net_j is computed for each input unit j by inverting the activation function. Since

$$\begin{aligned} o_j &= f(net_j) \\ &= \tanh(net_j) \end{aligned}$$

net_j can be computed by

$$\begin{aligned} net_j &= f^{-1}(o_j) \\ &= \operatorname{atanh}(o_j) \end{aligned}$$

Gradient descent on the error signal δ_j can be employed to alter net_j in order to minimize the difference between the actual and the nearest ideal output vectors using the equation

$$\Delta net_j = c \cdot \delta_j$$

where c is a small constant acting as the step size in input space. Altering a hypothetical net input to the input units during the gradient descent process, instead of the input unit activation levels directly, ensures that the activation level of the input units will remain within the -1 to +1 range of the hyperbolic tangent activation function.

So the steps in network inversion are:

1. Forward propagate activation from input to output units.

2. Compute error by comparing the actual output pattern with the nearest ideal output pattern.

3. Backward propagate this error from output to input units.

4. Alter the input unit activations to reduce the distance between the actual output and the nearest ideal output.

These four steps are repeated for a fixed number of steps, or until the difference between the actual and the nearest ideal output falls below a fixed threshold. At the end of this gradient descent procedure, the input vector in some sense represents the best input pattern for producing the target vector. This ideal input can then be compared with the actual input to measure the familiarity of the situation.

There is one significant problem with using network inversion for estimating the familiarity of an input pattern. In almost all task domains, the mapping from inputs to outputs is many-to-one, with many input patterns resulting in the same correct output response. In the domain of autonomous navigation for example, there are many road position/orientation combinations which should result in the same steering direction. The same is true in classification tasks such as speech or character recognition, where many different input exemplars are members of the same class. The network inversion technique will not produce a single "clean" example from the target class, but instead will produce a conglomeration or superposition of all the class members. This phenomenon is illustrated in Figures 9.2 and 9.3. For illustration purposes, the network in Figure 9.3 was trained on only the two artificial two-lane road images shown in Figure 9.2. Each depicted the road at a different position and orientation, but both required a straight-ahead steering direction. Because of the identical response required for each, the network does not learn to distinguish between them. Instead, each of the hidden units learns to respond identically to each. As a result, when presented with one of the training images, the reconstructed input produced by the network inversion method contains a superposition of both training examples.

Linden and Kindermann [Linden & Kindermann, 1989] exploited the superposition property of network inversion to analyze networks trained for character recognition. By thresholding the superimposed image created by network inversion, they were able to extract the essential or common features present in all

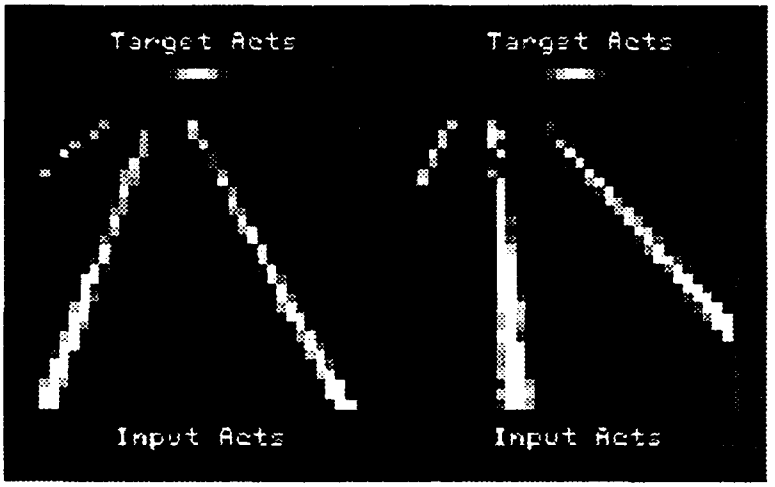


Figure 9.2: Two artificial road images used to train the network displayed in Figure 9.3. The roads have different positions and orientations, but they both require a straight-ahead steering direction.

examples of a particular character. But for accurate input reconstruction, this superposition of features is not desired, since the actual input should not be required to embody the conglomeration of all the features of a class's members in order for it to be considered a familiar member of that class. Instead, the algorithm should produce a pattern representing a "clean" single example of the class, and more specifically a single example which is as close to the actual input as possible. The actual input could then be compared with this "clean" nearby class member to determine how typical or familiar an example of the class it is.

9.3 Backdriving the Hidden Units

Part of network inversion's problem with superposition stems from the fact that it tries to reconstruct an ideal input pattern starting with an ideal output pattern. A better method would reconstruct the nearest familiar input pattern using the actual hidden representation formed by forward propagating the input pattern. This would enable more specific and distinct reconstructions of inputs which have different hidden representations, but which require the same output response.

This approach requires a different computation strategy than network inver-

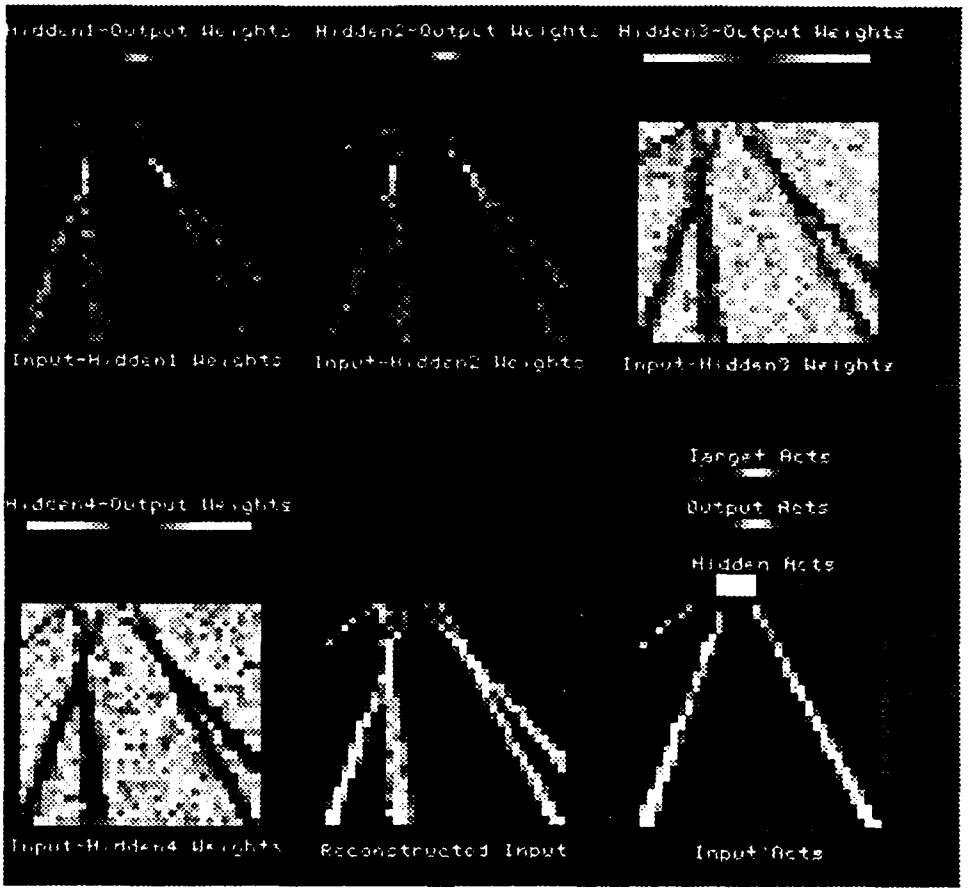


Figure 9.3: A network trained on two different images requiring the same steering direction. When presented with one of the images, the network inversion method produces a reconstructed input which is a superposition of both training inputs.

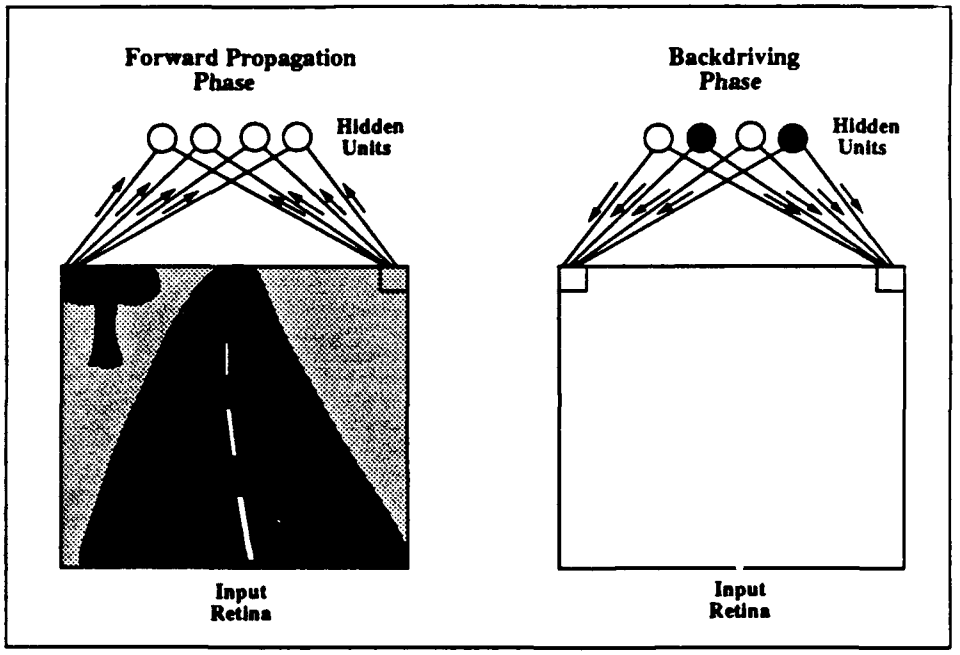


Figure 9.4: The two steps involved in reconstructing the input by backdriving the internal representation. In the first step, activation is propagated from the input layer to the hidden layer. In the second step, the hidden unit activations are frozen activation is propagated from then hidden layer back to the input layer.

sion, since the hidden representation provides no quantifiable error signal to be back-propagated for input reconstruction. I have developed an alternative technique that involves "backdriving" the network. The idea is to first present the network with an image and propagate activation forward from the input layer to the hidden layer. The hidden activation levels are then frozen and activation is propagate in the opposite direction, from the hidden layer to the input layer. This two step process is illustrated in Figure 9.4. Mathematically, the backdriving phase can be described as follows:

$$r_i = f\left(\sum_h w_{ih} o_h\right)$$

where r_i is the reconstructed activation for input unit i, f is the hyperbolic tangent activation function, w_{ih} is the weight of the connection from input unit i to hidden unit h, and o_h is the activation level of hidden unit h. By turning the network around in this manner, active hidden units will excite those input units which normally excite them, and inhibit input units which would normally inhibit them. Backdriving the first layer of weights reconstructs the input image which would produce the current hidden activation pattern.

Visual inspection of the actual and reconstructed input images demonstrates that the degree of resemblance between them is a good indication of the actual input's familiarity. Figure 9.5 shows the input image, network response, and reconstructed input at the three points along the stretch of road used to demonstrate OARE in the previous chapter. When presented with the image on the left, which closely resembles patterns from training set, the network's reconstructed image closely resembles the actual input. In novel situations, like the fork image and the two-lane road image shown in the other two columns of Figure 9.5, the reconstructed image bears much less resemblance to the original input, indicating the network's confusion.

To be useful, the visually apparent tendency of confusing input images to result in high reconstruction error must be quantified. In particular, the reconstruction error measure should be chosen so as to maximize its correlation with the network's steering error. In other words, the reconstruction error should be high on those inputs where the network is likely to make a mistake. Quantitatively capturing this visually apparent relationship has proven difficult. The problem is that the input image and reconstructed image have very differ means and variances, making a straight pixel-by-pixel distance metric such as Euclidean distance a poor measure of similarity between the images.

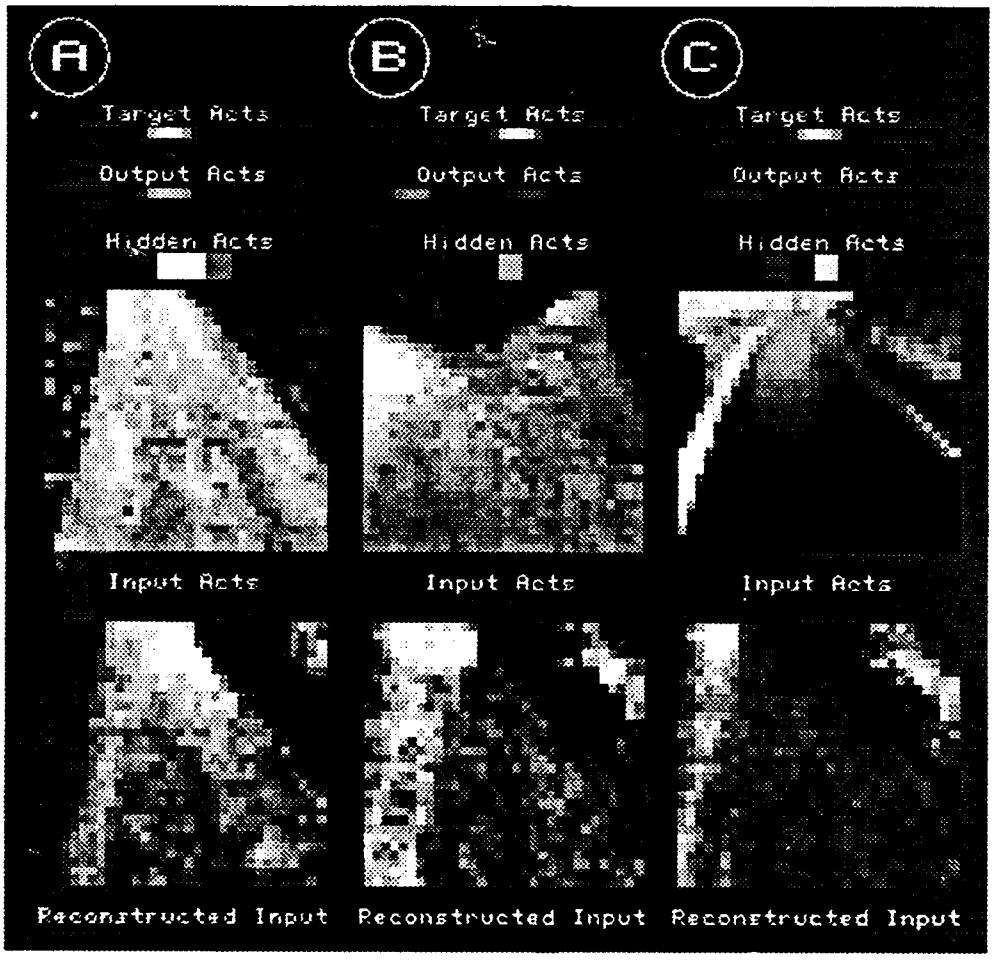


Figure 9.5: Input image, network response, and reconstructed input created by backdriving the input-to-hidden weights on three road images.

One method I have employed to better capture the degree of similarity between in the input and reconstructed images is to normalize the mean and variance of the reconstructed image to equal that of the input image. After this normalization step, the Euclidean distance between the two images provides a reliable measure of the images' similarity. A more straightforward technique for determining the similarity of two images with different means and variances is to compute the correlation coefficient between them.

The correlation coefficient $\rho(\mathbf{X}, \mathbf{Y})$ between two sets of numbers $\mathbf{X}$ and $\mathbf{Y}$, in this case representing image pixel values, is defined to be:

$$\rho(\mathbf{X}, \mathbf{Y}) = \frac{\overline{\mathbf{XY}} - \overline{\mathbf{X}} \cdot \overline{\mathbf{Y}}}{\sigma_x \sigma_y}$$

where $\overline{\mathbf{X}}$ and $\overline{\mathbf{Y}}$ are the means of each set, $\overline{\mathbf{XY}}$ is the mean of the set formed by the element-wise product of the two sets, and σ_x and σ_y represent the standard deviations of each set.

The correlation coefficient between the actual and reconstructed images has three useful properties. First, it is simple to compute. Second, like the normalization method described above, it compensates for the differences in mean and variance between the two images. Finally, it is bounded between -1.0 and +1.0, with -1.0 signifying a perfect inverse correlation, 0.0 signifying no correlation (i.e., independence), and +1.0 signifying perfect correlation between the images. Because of these useful properties, the correlation coefficient is the measure of similarity used throughout the remainder of this chapter. In order to compute the reconstruction error from this similarity measure, the correlation coefficient is subtracted from 1.0. This ensures that the reconstruction error will range between 0.0 and 2.0, and will increase as the similarity between the input and reconstructed images decreases.

The reconstruction error computed in this manner is a good indicator of network reliability on real images, as illustrated by the graph in Figure 9.6. It shows the steering error versus reconstruction error for the same network on the same stretch of road used to demonstrate OARE in the previous chapter. The 0.76 correlation coefficient between steering error and reconstruction error on this stretch of road is somewhat lower than the 0.87 correlation coefficient between steering error and output appearance error. One possible explanation for this lower correlation is that the reconstruction procedure is flagging some images as atypical for which the network is able to correctly generalize the steering direction. This is almost

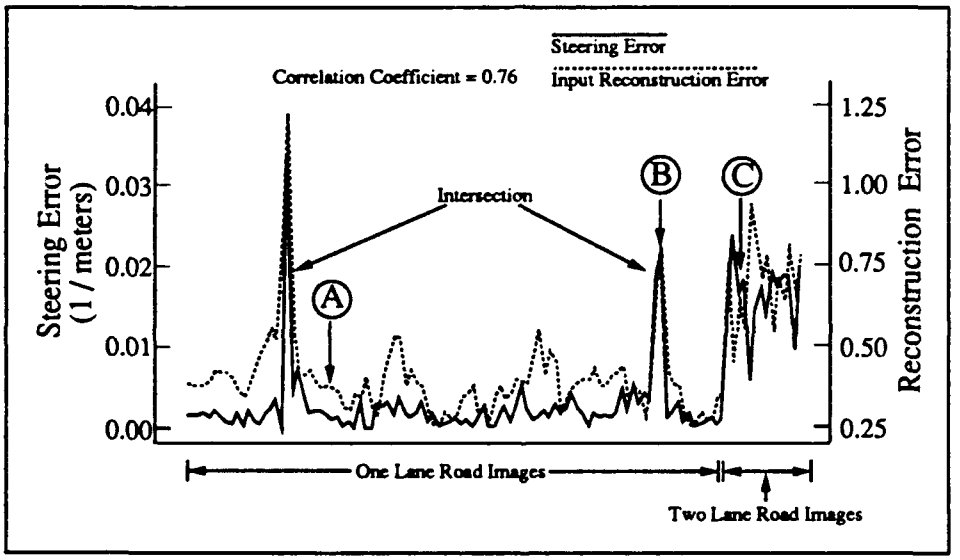


Figure 9.6: Reconstruction error obtained by backdriving the input-to-hidden weights versus network steering error over a stretch 500 meter of single and two-lane road.

certainly the case for the images in the middle of the graph in Figure 9.5, where the network is steering correctly but the reconstruction error is elevated. In fact, the first sizable peak after the point labeled A in Figure 9.6 corresponds to a left turn sharper than most in the training set. The network was able to choose the correct steering direction on that section of road, but the reconstruction error is indicating that something about that stretch is atypical.

A more theoretical problem which may also contribute to the lower accuracy of this reliability estimation method is the fact that backdriving the internal representation does not entirely solve the superposition problem described earlier. Input patterns requiring the same output response but with different hidden representations will no longer be superimposed in the reconstructed input. But if the network does not distinguish between input patterns in its internal representation, as was the case for the network trained on two straight-ahead images in Figure 9.2, the result of backdriving the input-to-hidden weights will still be an image in which the input patterns are superimposed.

The only way to avoid the superposition problem in input reconstruction is to force the network to develop different internal representations for distinct input patterns from the same class. The next section describes a technique for ensuring this discrimination, and gives the results obtained by using it.

9.4 Autoencoding the Input

One way to ensure that the network explicitly represents specific features from the current input in its hidden representation is to force the network to recreate the input in its output. An architecture which accomplishes this is shown in Figure 9.7. It contains an extra group of output units, with the same dimensions as the input image and fully connected to all the hidden units. The desired activation pattern for this new set of output units is identical to the input pattern.

Forcing the network to auto-encode its input as well as produce the correct steering direction makes the learning task more difficult. But it has two advantages for input reconstruction reliability estimation. First, unlike the two previous input reconstruction methods, it requires no extra computation after the forward pass through the network to produce the reconstructed image. The reconstructed input image is simply the activation pattern of the encoder output array.

More importantly, reproducing the input image in the output forces the network to develop different internal representations for distinct input patterns that require

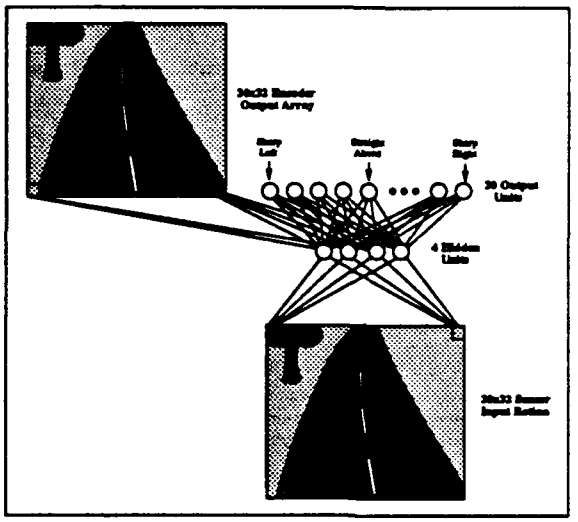


Figure 9.7: Network architecture augmented to include an encoder output array.

the same response. This is illustrated in Figure 9.8, which shows the architecture from Figure 9.7 trained on the same two straight-ahead artificial road images as the network in Figure 9.3. Like the network in Figure 9.3, this network devotes some of its hidden units (numbers 1 and 3) to encoding the correct steering direction. But unlike the network in Figure 9.3, this one also devotes two of its hidden units (numbers 2 and 4) to encoding specific details about features in the input image. Hidden units 2 and 4 are each excited by one of the two training patterns and inhibited by the other. Note that the weights to the steering output vector for hidden units 2 and 4 are near zero.

Using this pattern-specificity in the hidden representation, the network is able to accurately reproduce each of the two training patterns in the encoder output array (labeled "Reconstructed Input" in the Figure 9.8), without the superposition problem of the previous methods. But more importantly, on novel patterns like the image with the gap in the right lane marker, the reconstructed input accurately depicts the nearest training exemplar, and as a result "fills in" the gap in the lane marker. Both these points are illustrated by the "Difference" image in Figure 9.9. Each of the difference images was created by taking the point-wise absolute difference between the actual input and the reconstructed images. These differences were then thresholded in order to highlight the areas of high discrepancy. The large number of white pixels in the difference images for the two unfamiliar sit-

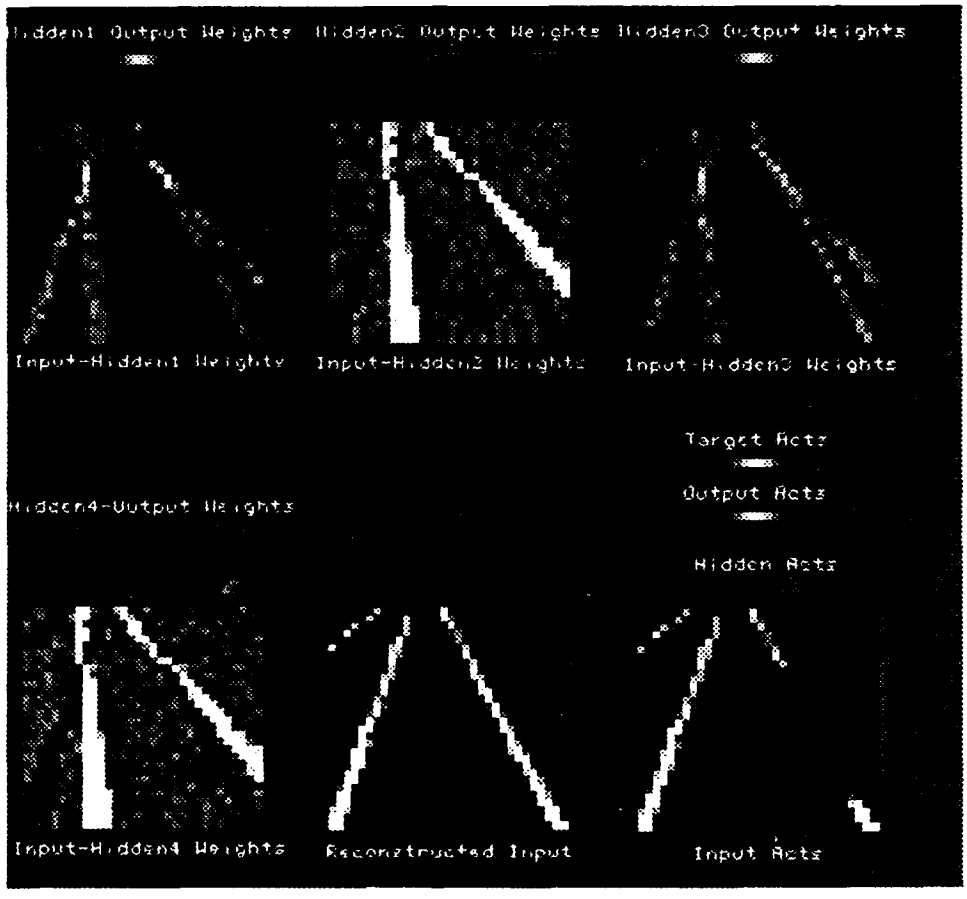


Figure 9.8: A network trained on two road images to produce the correct steering direction and autoencode the input.

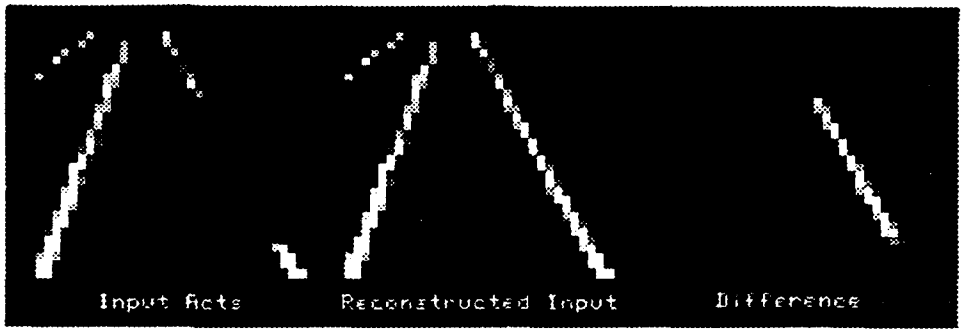


Figure 9.9: The actual input image, the reconstructed input image taken from the autoencoder output array, and the point-wise absolute difference between them. Notice that the difference image clearly points out the missing section in the right lane marker without superimposing the second road image.

uations illustrate that the network was not able to reconstruct the input images accurately. In addition, the well defined features in the difference image produced at the intersection suggests that it may be possible to reason about the specific aspects of the current input which are causing the network's confusion.

Eliminating the superposition problem results in a much higher correlation between reconstruction error and the novelty of the input image. As a result, the correlation between steering error and reconstruction error is significantly higher using autoencoder reconstruction than using either of the previous techniques. This strong correlation is illustrated in Figure 9.10, which is a graph of the steering error versus autoencoder reconstruction error of the network shown in Figure 9.7 trained on the road used previously. The input reconstruction error measure has a slightly higher correlation with steering error than does the output appearance error measure for the same network (0.92 vs. 0.90).

Figure 9.11 shows the actual input and the reconstructed input obtained from the autoencoder output array on the three images labeled A, B and C in Figure 9.10. The images labeled "Recon Error" represent the absolute value of the point-wise difference between the actual and reconstructed images. The brighter the pixel in the "Recon Error" image, the greater the difference. The network's confusion on the two atypical images is clearly demonstrated by the large discrepancy between the actual input and the reconstructed input. Note that the for in the image labeled B causes two roads to appear superimposed in the reconstructed image. This illustrates the specific reason for the network's confusion, namely that it sees two

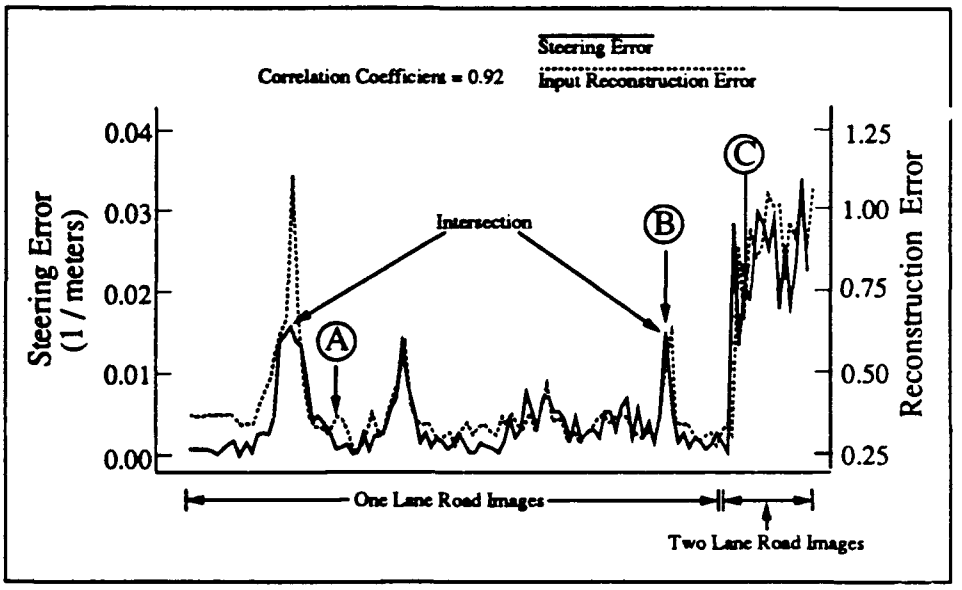


Figure 9.10: Reconstruction error obtained using autoencoder reconstruction versus network steering error over a stretch of one-lane and two-lane road.

possible roads it could follow.

9.5 Discussion

Intuitively, the effectiveness of input reconstruction reliable estimation stems from the fact that small number of hidden units in the network prevents it from faithfully encoding arbitrary input patterns. Instead, the hidden units learn to encode features present in the training images that are most important for the task. In the case of an autoencoder network like that shown in Figure 9.7, where the task is to reproduce the input image on the output, the definition of the most important features can be more precisely quantified using the relationship between back-propagation networks and the technique of principal component analysis.

Baldi and Hornik [1] have shown that if an autoencoder network with a single layer of N linear hidden units is trained with back-propagation, the activation levels of the hidden units will represent the first N principal components of the training set. The first N principal components are defined to be the N linear combinations

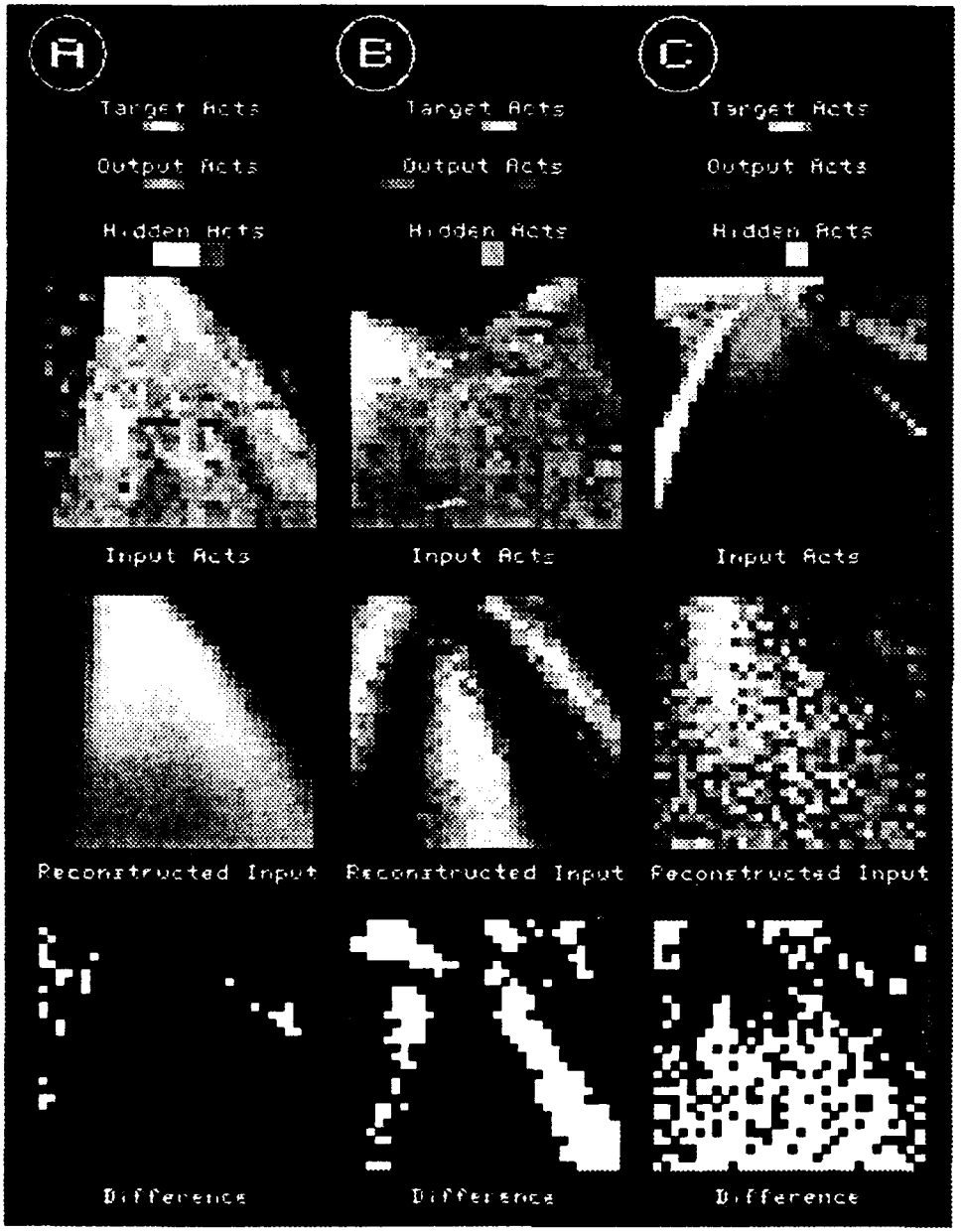


Figure 9.11: The actual input, the reconstructed input and the point-wise absolute difference between them on a road image similar to those in the training set (labeled A), and on two atypical images (labeled B and C).

of the input vector elements (i.e. pixels) that display the most variance over the training set. Because the principal components represent the dimensions along which the training examples varies most, it can be shown that using linear combinations of the principal components to represent the individual training patterns optimally preserves the information contained in the training set [3]. Specifically, the response functions developed by the linear hidden units represent the N linear basis functions that allow for the most accurate reconstruction of the training patterns. In other words, linear combinations of the N hidden activation levels can be used to reproduce the input patterns of the training set with lower sum squared error than combinations of any other N linear functions of the input pattern components.

The compressed representation developed by a linear autoencoder network is only optimal for encoding images from the same distribution as the training set. When presented with images very different from those in the training set, the reconstruction produced by a linear autoencoder network will not be as accurate. As I have demonstrated, this reconstruction error can be employed to estimate the likelihood of error in networks trained for autonomous driving.

The proof that the internal representation spans the principal components of the training set holds only when the hidden units employ a linear activation function. While this is not the case for the networks employed in this dissertation, Cottrell and Munro [2] have found empirically that autoencoder networks with a sigmoidal activation function develop hidden units that span the principal subspace of the training images, with some noise on the first principal component due to network non-linearity. This near optimality of the compressed representation developed by autoencoder networks may explain why the autoencoder reconstruction technique provides a better estimate of network reliability than either of the two alternative reconstruction techniques explored.

However reconstructing the input image using an autoencoder network has two disadvantages. First, it is the most computationally expensive of the three reconstruction techniques. Second, to achieve more accurate input reconstruction, the autoencoder reconstruction technique requires forcing the network to encode all input features, including potentially irrelevant ones. While this increased representation load on the hidden units has the potential to degrade network performance, this effect has not been observed in the tests I have conducted. Also, this potential problem could be avoided by having hidden units connected exclusively to one group of outputs or the other. This would prevent the representation developed for the autoencoder task from interfering with the representation developed for

the "normal" task. It remains to be seen if this decoupling will adversely affect IRRE's ability to predict network errors.

When compared with Output Appearance Reliability Estimation, IRRE has both advantages and disadvantages. OARE is simpler than any of the techniques for IRRE, but it has the potential to overestimate network reliability in novel situations. Input Reconstruction Reliability Estimation has the potential for the other extreme. Its assumption that the degree of novelty in the input is directly related to the network's reliability could result in underestimating the network's confidence when the novel input features are irrelevant. Perhaps the most accurate way to estimate network reliability would be to combine the predictions from the OARE and IRRE techniques.

A potential advantage of IRRE over OARE is the possibility of applying it to a wider variety of domains. The OARE technique is limited to tasks in which the correct response is determined by linear combination of many supporting input features. Estimating a network's reliability by measuring how faithfully the network is able to represent the input has no such application restrictions.

In addition, IRRE provides much more interesting insights into the internal processing of the network by depicting the network's interpretation of what it is seeing. Although I have not had a chance to exploit the added information provided by IRRE, its ability to specify the novel aspect of the current situation could potentially be used to reason about the appropriate response. In the missing lane marker example shown earlier, the IRRE technique could not only indicate that the current input is atypical, but also that its novelty results from a large gap in the right lane marker. This information could be employed by a symbolic reasoning module to identify the gap as the offramp it has been expecting.

Both OARE and IRRE have advantages over previous connectionist arbitration techniques. They both can estimate the reliability of isolated networks, instead of just the relative reliability of a group of networks. As a result, OARE and IRRE can predict performance errors in systems with only one network. In addition, OARE and IRRE have the ability to determine when none of a group of networks is reliable, and therefore a new network needs to be trained. In addition, both IRRE and OARE allow for the true modular construction of multi-network systems since new networks can be added incrementally, without the need to retrain the integrating structure.

These connectionist arbitration techniques also have distinct advantages over the rule-based arbitration techniques described in Chapter 7. Unlike symbolic techniques for multi-network integration, they provide a means of smoothly com-

bining responses from many networks based on confidence measures computed by the networks themselves. They make ALVINN less reliant on detailed and accurate high level knowledge of the environment, which is hard to obtain. In its place, OARE and IRRE substitute simple consistency checks between the network's response and some ideal, or between the network's internal representation and the actual input.

Of course OARE and IRRE cannot entirely replace symbolic reasoning techniques, but they can greatly augment them. This was demonstrated when OARE was used in conjunction with a coarse map to follow high level instructions while staying on the road. IRRE has the potential to increase the synergy between symbolic and connectionist processing by providing more detailed information about the novel aspects of the current situation.

Chapter 10

Other Applications - The SM^2

The techniques described in this dissertation are applicable in domains other than autonomous driving. In this chapter I present results which demonstrate this flexibility. I describe the application of these methods to the task of precisely guiding a robot called the Self Mobile Space Manipulator (SM^2), which is designed to walk on the exterior of space station Freedom [Brown, Friedman & Kanade, 1990, Ueno, Xu & Brown, 1990]. While the SM^2 is a very different type of robot, requiring very different guidance signals, the techniques developed for ALVINN require only minor modifications for this domain. The ease with which these techniques can be adapted to a new domain underscores the point made in the next chapter, that the learning power of artificial neural networks can effectively eliminate much of the difficulty involved in developing robust vision-based autonomous guidance systems.

10.1 The Task

Space is a dangerous environment for people. To reduce this danger in the construction and maintenance of the space station Freedom, a number of robot systems are under development. One of those robots, called the Self Mobile Space Manipulator (SM^2), is being designed at Carnegie Mellon University to perform visual inspection, transportation of parts and light construction tasks (see Figure 10.2). The SM^2, shown in Figure 10.1, is a two-legged robot capable of rapid walking along the outside of the space station. Grippers at the tip of each leg screw into threaded holes in the nodes where support struts join together.

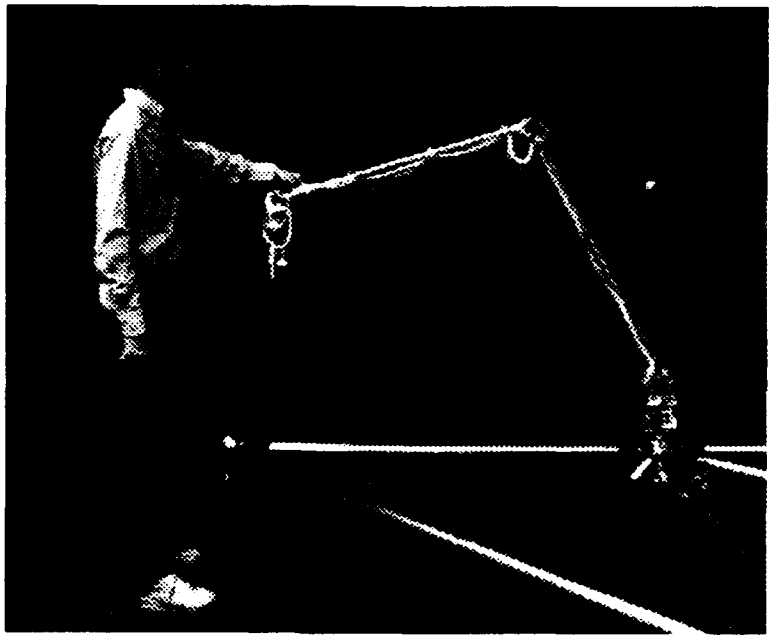


Figure 10.1: Image of the Self Mobile Space Manipulator.

Locomotion is achieved by alternately unscrewing one gripper from its anchor hole and swinging it around to screw into the next hole (see Figures 10.2 and 10.3).

In order build a compact, power-efficient robot capable of fast walking, the robot's mass has been kept to a minimum by using lightweight aluminum legs. As a result of the flexibility in these legs, and the long distance between adjacent anchor holes (15 feet on the space station, 5 feet on the 1/3 scale CMU testbed model), it is difficult to consistently position the gripper of the robot within the required 0.25 inches of the anchor hole for reliable insertion using only measured joint angles. In other words, sensor feedback is required for the precise positioning of the gripper. To facilitate this feedback, small monochrome video cameras are attached to the robot's grippers as illustrated in Figure 10.3. The next section describes the artificial neural network designed to guide the robot using these video cameras.

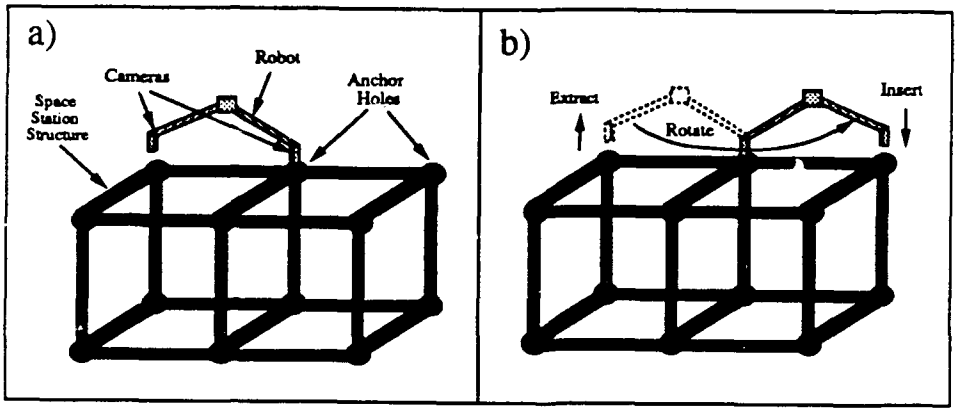


Figure 10.2: Schematic of the SM²: a) Robot and space station structure; b) Locomotion.

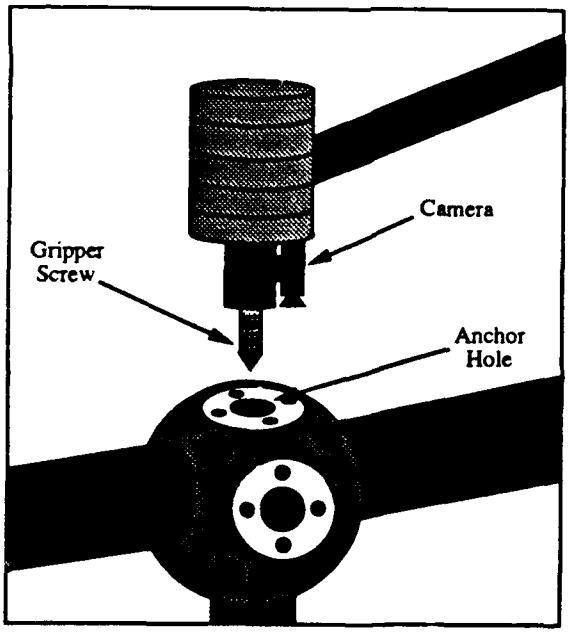


Figure 10.3: A schematic closeup of the gripper on the SM².

10.2 Network Architecture

Given an image coming from the camera at the gripper, the network is required to provide the two dimensional displacement of the gripper screw relative to the anchor hole. To accomplish this mapping, a neural network architecture very similar to that described for autonomous driving is used. Specifically, the input layer consists of a 24x20 two dimensional "retina" which receives input from the gripper's video camera. The activation level of each unit in the input retina represents the grey scale intensity of the corresponding pixel in the low resolution video image coming from the camera. Each unit in the input retina is fully connected to a layer of five hidden units which are in turn fully connected to two vectors of 20 output units (See Figure 10.4). The first output vector is a linear representation of the displacement of the gripper relative to the anchor hole in the X dimension, ranging from -1.25 inches for the leftmost output unit to +1.25 inches for the rightmost output unit. The second vector of output units is identical to the first except it represents displacement in the Y dimension.

To control the precise positioning of the robot's gripper once it is in the vicinity of the anchor hole, a video image from the appropriate gripper camera is projected onto the input layer. After completing a forward pass through the network, X and Y displacements are read off the output vectors. The technique described in Chapter 2 for interpolating the network's precise response by finding the best fit gaussian to an output vector is used to determine the network's displacement response in each dimension. These displacements are then converted to joint torques by the robot's PID controller in order to move the gripper over the anchor hole. Once the network has indicated that the gripper is within the 0.25 inch "threshold radius" of the anchor hole for over one second, the gripper is considered to be stably positioned over the hole and the controller lowers the gripper to anchor the leg.

10.3 Network Training and Performance

Like the networks used for autonomous driving, the network to control the SM^2 is trained using back-propagation. Also like its road following counterparts, this network for the SM^2 requires a relatively large and varied set of training examples in order to develop a general representation. However acquiring training examples is more difficult because there is no readily available teaching signal. To provide the precise X-Y gripper displacement measures required by the network, a special

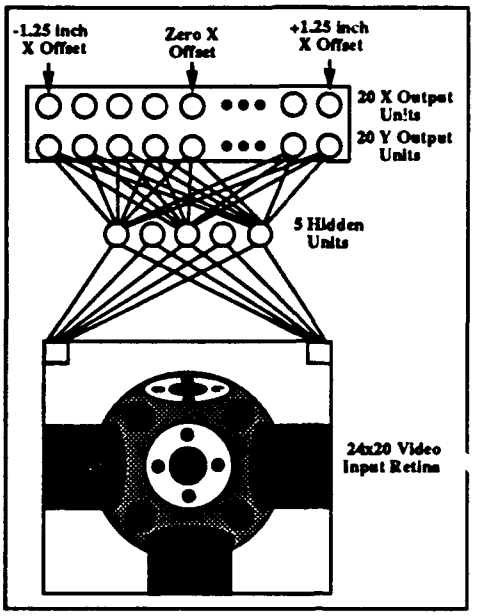


Figure 10.4: Architecture of network designed to control the SM².

training jig was constructed. The jig consisted of distance encoders which were temporarily attached to the robot gripper during the data collection phase (see Figure 10.5). 500 video images were taken while the gripper's position relative to the anchor hole was manually varied in three dimensions. Each of the collected images was automatically tagged with its corresponding X-Y offset.

During training, the back-propagation algorithm uses these input-output pairings as examples of the mapping to be performed. After approximately 100 presentations of the exemplars, the network learns to use image features to accurately determine the gripper displacement to within 0.1 inches. Figure 10.6 illustrates the evolution of the weights projecting into the five hidden units from the video retina at four different times during training. Prior to training, the network's connections are random, as illustrated by the unstructured distribution of positive weights (light squares) and negative weights (dark squares) at epoch 0 in Figure 10.6. As training progresses, Figure 10.6 shows the weights to the hidden units evolving to pick out important image features. The network uses the position of these features to determine the gripper offset relative to the anchor hole. The data collection and training phases require a total of about 20 minutes running on

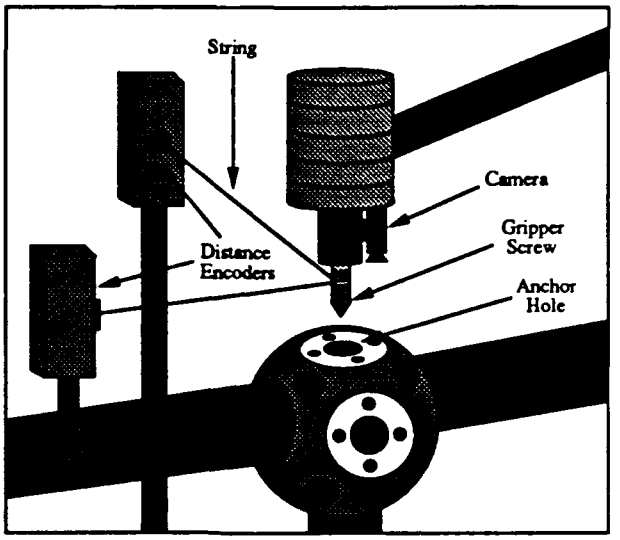


Figure 10.5: Distance encoders temporarily attached to robot provide accurate gripper displacement data during data collection.

a Sun-4 workstation.

Once trained the system is able to provide 15 precise and accurate gripper displacement values per second under a variety of laboratory lighting conditions. Figure 10.7 illustrates the low resolution video input to a trained network and the resulting network responses for X and Y gripper displacement. In the case of Figure 10.7, the robot gripper is actually centered over the anchor hole, although it does not appear to be, due to the camera's displacement relative to the gripper (see Figure 10.5). The network responds correctly in this situation by most strongly activating the centermost units of the two output vectors, indicating the gripper has zero offset in the X and Y dimensions. When compared with using no sensor feedback, this neural network method for precisely guiding the SM^2 resulted in a factor of five decrease in missed gripper insertions. Because of the success of the neural network approach, the non-connectionist gripper guidance implementations described in the next section were never tested on the robot itself, making it impossible to compare their performance.

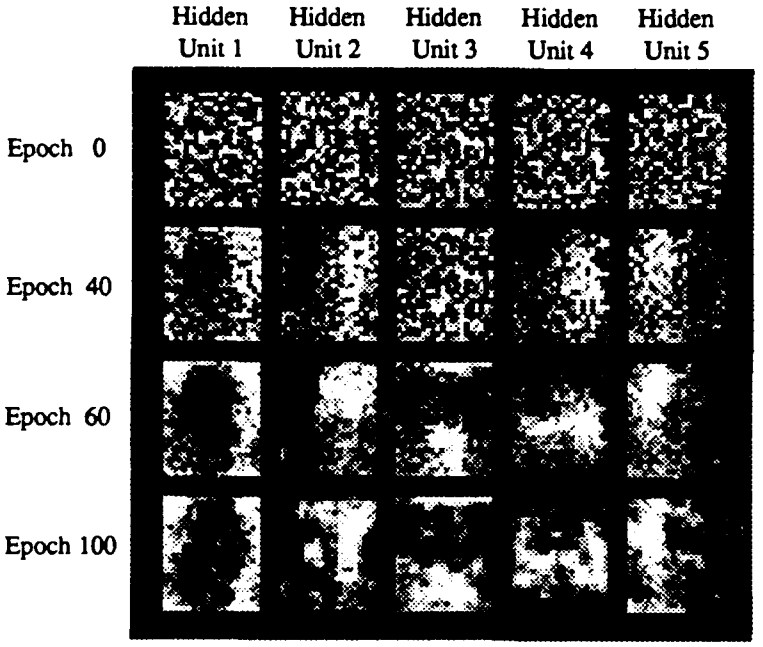


Figure 10.6: The weights projecting from the input retina to the five hidden units in the network at four points during training.

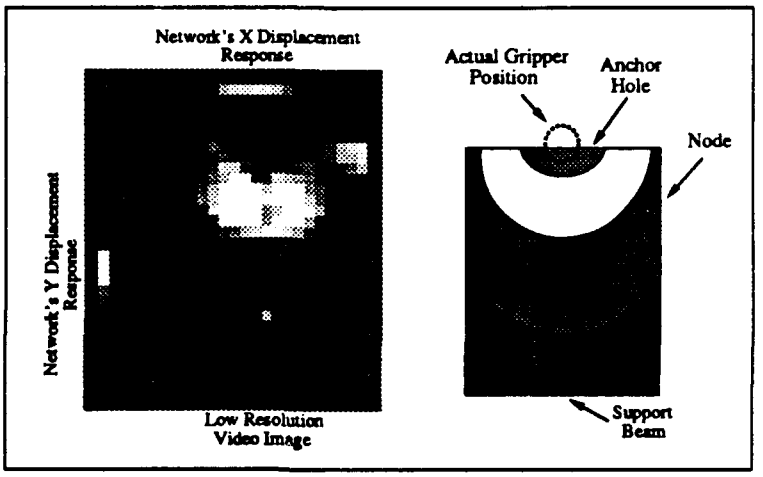


Figure 10.7: The low resolution video image provided as input to a trained network and the network's X-Y displacement responses. The gripper screw is actually centered over the hole in this image, as indicated in the schematic to the right. The anchor hole does not appear centered in the image is since the camera is offset from the gripper screw.

10.4 Discussion

The SM^2 example illustrates that a network similar to that employed for autonomous driving can precisely guide the foot placement of a walking robot. But what advantages do neural network learning methods have over traditional image processing techniques for performing this task?

First, the neural network system was very easy to develop. Once the camera and distance encoders were in place for data collection, it took under an hour to develop a working system from scratch. It will take even less time to develop a new network when a (highly probable) redesign of the space station structure occurs. This flexibility and ease of development results from the fact that the network does not rely on prespecified structural characteristics for its processing, but instead *learns* the features of the space station structure that are important for determining the gripper's relative position. If a redesign resulted in a different appearance, a new network could easily be trained to perform the task using the new structural features.

The two traditional image processing systems that have been designed for the same task [Simon, 1990] go one step further than requiring a fixed appearance for the space station structure. They require altering the space station's appearance by adding visual targets to make the image processing task more tractable. One system uses the black and white cameras currently on the robot and a white-on-black crosshair attached near each anchor hole. The system finds the crosshair in the image and uses its position to determine the gripper displacement. This system fails if the target is obscured or out of the camera's field of view. The second system uses a color camera and a red ring surrounding each anchor hole. Again, image processing techniques are used to find the ring and use its position to determine gripper displacement. This system is as robust as the neural network, but requires a more complex camera and, more importantly, alteration of the space station itself with a colored target. In addition, after choosing the target, the programmer of both these systems was required to develop algorithms for finding the target and determining the gripper offset from the target's size and location. In contrast, the neural network *learned* to perform the entire task from examples. Neural networks will be demonstrated to have similar advantages over previous hand-coded systems for autonomous driving in the next chapter.

Since the neural network was easy to develop and can learn to robustly provide accurate gripper displacement data without altering the space station's appearance, it is the vision system being employed in the development of the SM^2. In addition

to being a useful solution to a difficult problem, the neural network SM^2 guidance system illustrates that the vision-based neural connectionist techniques developed in this thesis are applicable to more than just a variety of driving tasks, but also to very different mobile robot guidance domains.

Chapter 11

Other Vision-based Robot Guidance Methods

In the last chapter, I demonstrated how the flexibility of the techniques described in this dissertation make them applicable to other vision-based robot guidance tasks. The techniques were also shown to have distinct advantages over more traditional, hand-coded machine vision algorithms for guiding the SM^2. In the first section of this chapter, I expand the comparison between the connectionist methods I've developed and non-learning techniques others have employed for autonomous navigation. As was the case in the walking robot domain, ALVINN is shown to have distinct advantages relative to hand-programmed systems for autonomous driving.

In the second part of the chapter, I compare the methods I've developed with other machine learning techniques for autonomous navigation. Some of these techniques are neural network based, while others use such methods as reinforcement learning, genetic algorithms, decision trees and nearest neighbor matching. Some have been tested by others in the domain of autonomous navigation, some I have tried myself, and some I only speculate about. The general conclusion is that other supervised connectionist learning algorithms could probably learn to drive as well as the back-propagation system presented in this dissertation, and could in fact exploit many of the techniques I've developed. But the other machine learning paradigms have significant shortcomings which would make it difficult to duplicate ALVINN's level of performance.

11.1 Non-learning Autonomous Driving Systems

Almost all of the previous vision-based outdoor autonomous navigation systems have been hand-programmed using the following three step procedure: First, the programmer determines which image features are important for navigating in the chosen domain. Then the programmer develops detectors (using statistical or symbolic techniques) to find these important features. Finally, the programmer develops an algorithm to determine the appropriate steering direction given the positions and orientations of the important image features.

11.1.1 Examples

A typical example of a system employing this three step procedure was the Martin Marietta VITS system [Turk et al., 1988]. This system assumed the road would appear as bright pixels in the image formed by subtracting the red color band from the blue color band. It thresholded this feature image and found the largest region of "road pixels". The borders of this region were back-projected to the ground plane and the vehicle was steered towards the center of this back-projected road region. Using this algorithm, the VITS system was able to drive at speeds of up to 20 km/hour on straight, obstacle-free stretches of a single-lane road. Its crude segmentation technique and reliance on a single sensor (video camera) limited its robustness in the domain of single-lane driving, and prevented it from tackling other domains such as multi-lane driving or obstacle avoidance. The Martin Marietta system was also not designed to handle intersections or to follow high level directions.

The SCARF system [Crisman & Thorpe, 1990] was similar to the VITS system in that it classified pixels in the image into road or non-road based on their color. However SCARF used a more sophisticated, probabilistic, adaptive color classification scheme to label pixels. SCARF then used a Hough transform [Ballard & Brown, 1982] to find the most probable road location, assuming the road appears as a trapezoid in the image. It then back-projected the road trapezoid to the ground plane and steered to keep the vehicle centered on this region and heading in the direction of the region's major axis. Its more sophisticated segmentation algorithm made SCARF much more robust then Martin Marietta's ALV system when driving in changing and noisy situations in the domain of single-lane road following. But like the Martin Marietta system, SCARF used a single sensor (video camera) and looked for a single image feature (a trapezoidal-shaped re-

gion of road pixels) to determine the direction to steer. This limitation prevented SCARF from being used in other domains, such as multi-lane road driving and obstacle avoidance. An interesting feature of SCARF missing in the VITS system was the modeling of intersections as overlapping trapezoids. This gave SCARF the ability to traverse intersections, at least on the single-lane CMU test path.

The Sidewalk II system [Goto et al., 1986] was a predecessor of the SCARF system, and hence used a more crude segmentation algorithm. However it did have interesting characteristics related to the ALVINN system. Unlike other previous systems, it fused input from a laser rangefinder and a video camera in order to distinguish between stairs, a ramp, and the surrounding grass. The Sidewalk II system also used an internal map of the CMU sidewalk network to plan a path through intersections. However the Sidewalk II system was limited to driving on a single terrain type (sidewalks) and, because of its crude segmentation algorithm, displayed limited reliability in that domain.

Instead of using color to classify pixels into road and non-road regions, a number of systems have employed edge detection methods to find road or lane boundaries. The General Motors Lanelok system [Kenue, 1989] used a Sobel edge operator [Sobel, 1970] and a Hough transform to detect the position of the lane markers on a multi-lane highway. The lane markers were assumed to fit a straight line, and their image position was used to control the vehicle.

Another system that employed edge detection was the VaMoRs system from the Universitat der Bunderswehr Munchen in Germany [Dickmanns & Zapp, 1987]. This system used local edge detectors to find the position of highway lane markers in various subregions of the image. A curve was then fitted to the detected points on the lane markers and a sophisticated control algorithm was employed to keep the vehicle in its lane. Because of its local feature detectors and parallel implementation, the VaMoRs system was capable of processing 13 images per second and driving at up to 60 miles/hour on the German Autobahn.

Edge detection systems like the Lanelok and VaMoRs systems assume important features will have strong edges associated with them in the image. On freshly-painted highways this is usually the case. However, on older highways with faded lane markers and cracked pavement, heavily-shadowed stretches of road, or roads without lane markings, important edges are often obscured or non-existent.

To avoid the difficulties that result from relying solely on edges, the YARF system used edge *and* color information to find lane markers on multi-lane highways [Kluge & Thorpe, 1990]. Like the VaMoRs system, it used local trackers to

find the position of important road features within small windows in the image. But unlike the VaMoRs system, YARF had a number of different hand-programmed feature trackers which used color information to more reliably locate features such as the yellow center line and the white line indicating the road edge. In addition, the YARF system reasons about the information received from its feature trackers to alter the trackers it uses as conditions changed. This is analogous to the capability of the ALVINN system to arbitrate between its multiple knowledge, described in Chapters 7, 8 and 9.

11.1.2 Comparison with ALVINN

In order to achieve greater reliability, the trend in autonomous navigation systems has been towards more complex, situation-specific processing. This not only makes system development more difficult, it also restricts the domains each of these systems is capable of handling. In contrast, ALVINN is able to *learn* for each of these domains what image features are important, how to detect them and how to use their position to steer the vehicle. As was demonstrated in Chapter 6, ALVINN's hidden unit representation varies depending on the driving situation. ALVINN learns to key off image features which correlate with the correct steering direction. When trained on multi-lane roads, the network develops hidden unit feature detectors for the lines painted on the road, while in single-lane driving situations, the detectors developed are sensitive to road edges and road-shaped regions of similar intensity in the image.

ALVINN's ability to quickly adapt to new scenarios is a significant advantage, and it has allowed ALVINN to drive in a wider variety of situations than any other autonomous navigation system. However when dealing with only minor changes in a driving situation, using neural networks can be a disadvantage. For instance, a new network would have to be trained to handle a road which is identical to a previous road but twice as wide. In contrast, a traditional machine vision program which had been hand-coded to look for features at particular locations could simply be told to look for those same features at a different location.

Another advantage ALVINN has over previous systems results from the simplicity of the connectionist approach. Determining the direction to steer from an image requires only a forward pass through the network (essentially two matrix-vector multiplies and some table-lookups to approximate the squashing function), resulting in faster processing and hence faster driving than is possible with other methods. As a result, using the same computing resources, ALVINN has been

able to drive the CMU autonomous vehicles 4 to 6 times faster than the situation-specific hand-coded vision systems in each of their domains of expertise.

Finally, since the network does not depend on finding a single feature in the image but instead subtly combines image features to determine the correct steering direction, its behavior is robust in situations where features are missing or obscured. For instance, if the road's center line is hidden by a passing car or harsh shadow, ALVINN will be less affected than a system that depends on just finding the center line, since ALVINN also uses cues from other images features like the road edges to determine the correct direction to steer.

However this ability to utilize subtle image features can be dangerous. This danger was demonstrated when ALVINN was trained to drive on a dirt road with a small but distinct ditch on its right side. The network had no problem learning and then driving autonomously in one direction, but when the vehicle was turned around, the network stayed on the road, but was erratic, swerving from one edge of the road to the other. After analyzing the network's hidden units, the reason for its difficulty became clear. It had developed detectors not only for the position of the road, but also for the position of the ditch on the right side during training. When tested in the opposite direction, the network was able to keep the vehicle on the road using its road detectors but was somewhat confused because the ditch it had learned to look for on the right side was now on the left. As a result of the network's ability to key on *any* image feature that correlates with steering direction, it is important that during training the network be presented with the entire range of situations it will later encounter, to prevent it from relying on local, coincidental features such as a ditch on one side. The technique of adding structured noise to the input (described in Chapter 4) helps to reduce the network's reliance on irrelevant image features.

11.2 Other Connectionist Navigation Systems

A few other previous systems have employed connectionist supervised learning techniques to do tasks related to outdoor autonomous navigation. One system build at Martin Marietta [Marra, Dunlay & Mathis, 1988] used a neural network to classify video image pixels into one of the categories dirt, grass, road or sky. The network was given a 16x16 window of color pixels taken from a full resolution video image, along with an encoding of the window's vertical position in the larger image. The network was trained to determine the category of that portion

of the image. By sequentially presenting each subregion of the larger image to the network and recording its classification, the entire image could be segmented into regions of dirt, grass, road and sky. The network was able to identify road pixels with relatively high accuracy, but was not designed to drive a vehicle. A similar system was built at British Aerospace [Wright, 1989]. This system used more sophisticated statistics about the region in question, such as its brightness variability, to train a network to classify regions. In both the Martin Marietta and the British Aerospace systems, training the network and classifying an entire image were too time-consuming for practical application to driving. In addition, to control a vehicle, each of these systems would require additional processing to determine the appropriate steering command from the segmented image. Like the non-learning systems, they would require a hand-programmed model of what roads look like, and therefore would suffer from the same situational dependence and brittleness.

Other connectionist autonomous navigation systems have employed simulated and/or simplified indoor environments. An interesting example from this category was built at TRW by Shepanski and Macy [Shepanski & Macy, 1987]. The system used a neural network to control a car in a simulated "arcade game" driving environment. The network received as input information such as the lane its vehicle currently occupied, the location of other cars and the curvature of the upcoming road. The network was trained using back-propagation to change speed and direction in order to stay on the road and avoid other cars. Unlike ALVINN, the system performed no perceptual processing, since it was provided with a simple, symbolic encoding of all the relevant features. However, like ALVINN, the system learned by watching people drive. In fact, the system adopted the driving style of its trainer. If it learned by watching a conservative driver, it would hardly ever pass. If it was taught by an aggressive driver, the network drove recklessly and tended to cut off other cars.

A number of other researchers have used techniques related to neural networks, such as potential fields [Tarassenko et al., 1991, Bachrach, 1991] or reinforcement learning [Sutton, 1990, Dayan, 1991, Thrun, 1991] to explore problems in planning and obstacle avoidance. Like the Shepanski and Macy system, these systems place little or no emphasis on visual perception, and hence would not be immediately applicable to outdoor navigation. Application of reinforcement learning techniques to autonomous outdoor driving would be particularly difficult because of the catastrophic result of errors. A network could not afford to drive haphazardly at the beginning and count on learning from its mistakes, since in outdoor driving

a single mistake could cause irreparable damage to the vehicle. An additional difficulty inherent in applying reinforcement learning to real world perception tasks is the credit assignment problem for high dimensional inputs. When the system is only told it has made a mistake after it has driven off the road, it faces the dual problems of determining which of its recent actions caused the failure, and which of the many input features in the situation where it made the mistake should it have treated differently. This credit assignment problem will appear again in the analysis of non-connectionist learning methods for autonomous navigation.

Another neural network like approach to mobile robot guidance is the work of Brooks [Brooks, 1986]. Brooks' robots are hand-programmed to perform simple behaviors like wall following and climbing over obstacles. Like in the ALVINN system, these simple behaviors can be linked together to create more sophisticated capabilities. The real innovation in Brooks' work is the use of small robots. When the robot is the size of a shoebox, it can afford to run into obstacles first and then determine how to get around them. Thus his robots require only relatively simple sensing and processing. In contrast, when the robot is a six ton truck traveling at 55 mph, it doesn't have the same luxury. It must look ahead and make very precise responses based on subtle sensor cues in order to avoid disaster. For tasks where a small robot is sufficient, Brooks' approach may work well. But for applications such as autonomous mail delivery system or the enhanced cruise control discussed in the next chapter, the robot needs to be large since it will be transporting large payloads over long distances.

11.3 Other Potential Connectionist Methods

There are many alternative connectionist approaches which could conceivably be employed for autonomous navigation. They can be divided into supervised and unsupervised learning algorithms. In the supervised category are a number of techniques which use the error surface's second derivative (or an approximation to it) to speed up gradient descent. Two prominent examples are the quickprop algorithm [Fahlman, 1988] and the conjugate gradient method [Rohwer, 1991]. While I have not tried these techniques for ALVINN, I see no reason why they would not work. However I don't expect they would provide much advantage, either in driving accuracy or in learning speed. One reason is that these alternative training algorithms do not produce qualitatively different networks than standard back-propagation. In addition, ALVINN's learning speed is not limited by back-

propagation, but by the time required to collect and transform the training images. To learn to drive accurately, ALVINN must be presented with a relatively varied set of training examples. Even with faster learning algorithms, this training phase would still require a person to drive the vehicle over a substantial stretch of road.

A number of supervised connectionist learning algorithms which differ substantially from standard back-propagation could potentially be employed to training ALVINN networks. One such alternative would be to use radial basis functions [Poggio & Girosi, 1990] instead of the standard sigmoid shaped activation function for the units in the network. Hidden units with radial activation functions divide the multi-dimensional input space into hyperellipsoid-shaped regions, instead of carving it up with soft hyperplanes as standard back-propagation does. Radial basis function networks, like networks with the standard sigmoid activation function, have been shown to be universal approximators [Baldi, 1991] (i.e., theoretically capable of approximating any reasonable function to any degree of precision). However, units with radial activation functions could have more trouble representing important image properties related to autonomous navigation than standard sigmoid units. The reason is that individual hidden units with radial activation functions behave very much like template matchers, being maximally activated by a particular input pattern and becoming less active as the input differs from the unit's ideal template. In contract, units with a sigmoid-shaped activation function do not behave like template matchers, since they can independently detect many features at different image locations and combine the results to determine its output. The problem with template matching and the related technique of nearest-neighbor matching, is discussed in the section comparing ALVINN with other non-connectionist learning techniques.

But before discussing non-connectionist methods, there are two additional classes of connectionist learning algorithms worth speculating about for autonomous navigation. One recently developed class of connectionist algorithms are those which modify the network architecture as learning progresses. Example of this approach are Cascade-Correlation [Fahlman & Lebiere, 1990] and meiosis networks [Hanson, 1990], which add hidden units to the network as learning progresses. In the tests conducted to date with this algorithm, the entire training set has been present throughout the learning process. To the best of my knowledge, no experiments have been conducted using Cascade-Correlation in which the training set varies over time. The dynamic nature of the training set when learning to drive in real time could potentially cause problems for Cascade-Correlation for the following reason. The algorithm depends on teaching a small network to do

the best it can on the training set, and then adding new hidden units to eliminate residual errors. With the training set constantly changing, it could be difficult to determine when the current architecture is doing the best it can and therefore when to add additional hidden units. However this problem might be alleviated using a modified version of the pattern buffering technique currently employed in training on-the-fly.

A final class of connectionist algorithms with potential application to autonomous navigation are the unsupervised learning methods. These allow the network to discover its own regularities in the input, without the guidance of an external teacher specifying a desired output. Two subclasses of unsupervised learning methods are variants on clustering algorithms [Rumelhart & Zipser, 1986, Grossberg, 1987, Kohonen, 1990, Fukushima, 1988] and neural network techniques for performing principal component analysis [Oja, 1989, Linsker, 1989]. Clustering algorithms work well when the goal is to find a limited number of prototypes in a set of high-dimensional data. Principal component analysis seeks to achieve maximal data compression while retaining as much information as possible about the input. These unsupervised techniques frequently find interesting features in the input, but alone cannot produce a particular desired response (e.g., the correct steering direction). Exciting recent results have been achieved by combining supervised and unsupervised learning methods in the same network [Moody & Darken, 1989]. Moody and Darken trained the initial layer of weights to pick out interesting input features using an unsupervised algorithm and trained the subsequent layer to perform a specific task using back-propagation. They found this hybrid technique resulted in faster learning and comparable performance to supervised training methods on the difficult tasks of phoneme recognition and chaotic time series prediction. This type of hybrid approach could also have applications to vision-based autonomous navigation.

11.4 Other Machine Learning Techniques

A number of other techniques exist in the machine learning literature which are not based on the connectionist paradigm. They too should be considered as potential methods for autonomous navigation. Perhaps the oldest are the nearest neighbor matching algorithms [MacQueen, 1967]. The idea is to "learn" the task by recording samples of the inputs along with the correct response. Then, to perform the task, an actual input is compared with each stored example to find one

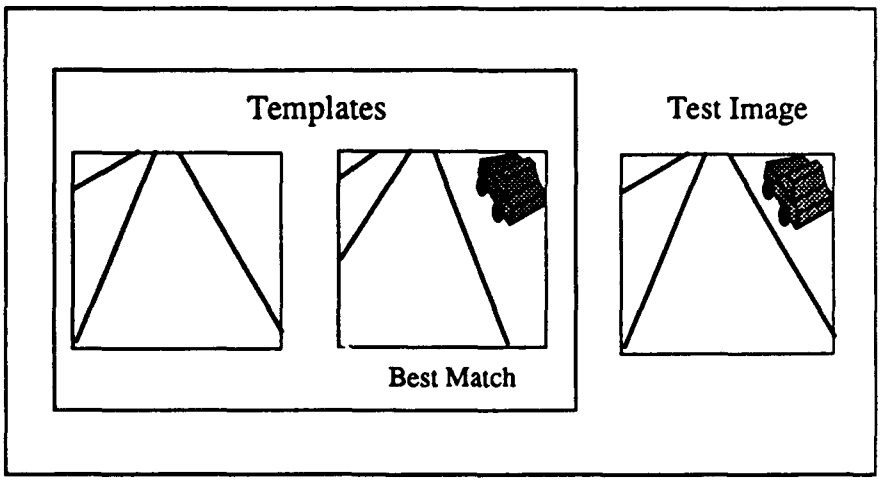


Figure 11.1: A example of the problem with nearest neighbor matching. The template which should match the current image is the one on the left. However an irrelevant feature in the periphery, a car, causes the template on right to match most closely.

or more close matches using some distance metric (such as Euclidean distance). The response associated with the closest match, or the average response from a few close matches, is assumed to be the correct response for the current input.

I have tried this technique for autonomous navigation and found it to have significant problems. Conceptually, the problem can be understood from the simple example in Figure 11.1. In this example, there are two templates of simulated two-lane road images, presumably collected earlier while a person drove. The first requires a straight-ahead steering command, while the second requires a left turn since the vehicle is slightly off the right side of the road. The test image presented to the system is shown on the right side of Figure 11.1. The position and orientation of the road markers in the test image are identical to those of the first template, so one would hope the matching algorithm would determine it to be the nearest neighbor. However, because of the limited size of the lane markers in the image, the closest matching template is actually the second template, in which a large irrelevant feature, the car on the right side of the road, matches the current image closely. The back-propagation training algorithm, particularly when augmented with the structured noise training technique described in Chapter 4, teaches the network to weight important features like lane markers

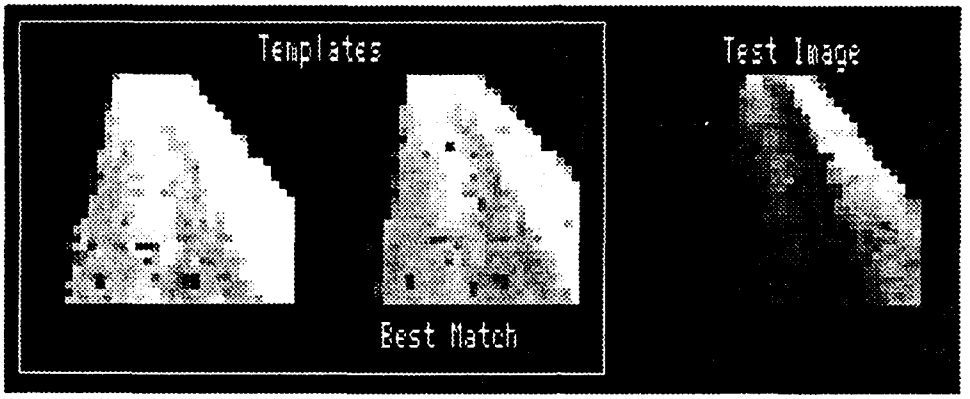


Figure 11.2: A example of nearest neighbor matching making a mistake on real images. The template on the right matches the test image better than the template on the left, despite the fact that the left template and the test image depict the exact same scene, but under different lighting conditions.

near the center of the road more heavily than irrelevant features like cars in the periphery of the image. In contrast, nearest neighbor techniques and other template matching algorithms *treat all input pixels equally*. It might be possible to improve nearest neighbor matching in this type of situation by adding structured noise to the templates, or alternatively developing a weighting function which discounts the closeness of match of peripheral pixels. However, the task of appropriately weighting input features requires more than just ignoring peripheral pixels, as is illustrated in the following experiment.

Images of a single-lane stretch of road were collected under two different weather and lighting conditions. Under the first condition, illustrated in the two images labeled templates in Figure 11.2, the pavement was dry, resulting in fairly uniform intensities for all the road pixels in the templates. Under the second condition, the pavement was wet, resulting in specular reflection off the road surface. This specular reflection resulted in the right side of the road being significantly brighter than the left, as shown in the image labeled Test Image in Figure 11.2. The test image was taken with the vehicle in the exact same position and orientation as in the left template. However because of the intensity gradient across the test image, the template on the right is actually a better match to the test image than the one on the left. This mismatching of templates causes the system to steer more sharply right than it should.

As in the example with the car in the periphery, the problem here is that all pixels are given equal weight when determining the nearest template. In many cases, there are certain pixels which are much more important than others for determining the correct steering direction. In this specific case, the pixels along sharp image boundaries are more important than pixels in a region's interior. ALVINN is able to perform well because it develops detectors that emphasize the positions of relatively localized image features, like region boundaries, instead of simply computing global image similarity.

It is conceivable that a heuristic could be developed to solve this problem with nearest neighbor matching. In fact, a neural network with a single hidden layer of units can in some sense be considered just such a solution. For instance, it was shown in Chapter 6 that ALVINN's hidden units act as filters or templates for combinations of input features at particular positions and orientations. The network learns from examples to weight the importance of various image features in order to perform the task correctly. A more explicit, hand-generated weighting of various features might be employed to encourage a traditional nearest neighbor matching algorithm to pay more attention to particular features. Towards this end, the techniques of transforming the live training images and adding structured noise to the training data could serve the same purpose for a nearest neighbor matching system as they do for ALVINN. Specifically, they could increase the variety in the template set and hence enhance the nearest neighbor matching system's ability to handle situations not explicit represented in the live data.

But like with the earlier techniques explored in this chapter, I do not expect such a modified nearest neighbor matching algorithm to outperform ALVINN for the following reasons. The necessary improvements to mitigate the effects of matching irrelevant features would all require increasing the cost of determining the best match, either by increasing the number of templates to match against, or by making the matching process itself more complex. This would slow down processing and make real-time driving difficult to achieve. The connectionist approach can be thought of as a technique for distilling into a compact form all the relevant knowledge from the training set that would normally be used in a nearest neighbor matching system. ALVINN's distributed representation, with single hidden units frequently acting as templates for roads at different positions and orientations, allows it to determine the correct steering direction with just the few thousand arithmetic operations required to forward propagate through the network. The equivalent computation would be much more expensive if hundreds of templates had to be compared with the current image to find the closest match, even using

cost saving search techniques like k-d trees [Friedman, Bentley & Finkel, 1977]. There is a penalty associated with distilling the information contained in those hundreds of images into the weights of a neural network, namely the additional time required to train the network. But this cost is small, particularly since the training on-the-fly technique allows much of it to be incurred while the training images are being collected.

Another commonly employed machine learning tool used for classification is decision trees [Quinlan, 1986, Breiman et al., 1984]. This paradigm suffers from the opposite problem that plagues nearest neighbor matching. Decision trees have no difficulty learning to weight certain features more than others. Decision tree algorithms classify patterns by making a sequence of decisions based on the value of individual attributes within the pattern. The sequence of decisions, and hence the input attributes upon which those decisions are based, have a natural ordering of importance based on the amount of information provided by each feature. The problem is that decision trees base their branching on *single values* from the input pattern; there is no easy way of represent larger scale image features like edges or regions which are composed of many pixel values[1]. As a result, a successful application of decision trees to vision-based autonomous navigation would require large, deep trees. These trees probably wouldn't generalize well to new inputs unless they were built with a very large number of training patterns.

Genetic algorithms [Holland, 1975, Goldberg, 1989] suffer from the same problem as decision trees, namely they have difficulty representing higher level structure in the input. Recent work in combining genetic algorithms with supervised neural network learning has the potential to correct this shortcoming [Keesing & Stork, 1991]. The idea is to simultaneously train a number of networks with different architectures, initial weights or learning rate parameters, and choose to replicate (with some small modifications) only those which perform well. For autonomous navigation, the large network size and real time training constraints currently make this type of iterative procedure prohibitively expensive.

11.5 Discussion

Because of its ability to learn to drive from examples, ALVINN has two distinct advantages over previous autonomous navigation systems. First, it can drive more

[1] One might say that decision trees can't see the forest for the trees.

reliably in a wide variety of situations because it learns on its own which image features are important and how to use them to drive. This contrasts with previous systems, which have relied on the judgement and programming of people, who may have trouble explicitly converting their driving knowledge into a form the computer can understand. By circumventing the tedious programming phase, ALVINN also demonstrates the novel ability to quickly adapt to new driving situations.

Other machine learning techniques have the potential to exploit the same advantages ALVINN possesses for vision-based autonomous driving. But in my opinion, there are no alternative learning methods that clearly promise better performance than the artificial neural network-based ALVINN system. Those that have the potential to succeed would require many of the techniques developed in this dissertation, such as the image transformation scheme for augmenting the training set, in order to match ALVINN's performance level.

Chapter 12

Conclusion

In this dissertation, I have developed techniques which enable artificial neural networks to guide mobile robots using visual input. By exploiting their ability to adapt to new situations, I have shown that artificial neural networks are capable of reliably controlling a number of different mobile robots using a variety of imaging sensors in many different circumstances. The ALVINN system has driven two different wheeled vehicles using video and laser sensor input for distances of up to 21.2 miles at speeds up to 55 miles/hour. ALVINN has driven in a wider variety of situations than any previous autonomous navigation system, including following single and multi-lane roads, avoiding obstacles and tracking prominent terrain contours such as rows of parked cars.

The flexibility of the techniques I have developed allow them to be applied to mobile robot domains other than driving. One such alternative domain is guiding the foot placement of a walking robot. The connectionist methods described in this thesis resulted in a fivefold increase in foot placement accuracy when used to control an inspection robot for the exterior of the space station.

12.1 Contributions

In developing neural network techniques for autonomous robot guidance, I have addressed a number of previously unexplored difficulties in applying connectionist methods to real world problems. First among these are difficulties resulting from restrictions on the training data. Due to time constraints or other logistical complications, sufficient training data is hard to acquire in many real world problems.

For autonomous driving, this problem manifests itself in two ways. First, since the human trainer drives quite accurately, the training data contains no examples of recovering from errors. The image transformation schemed developed for training on-the-fly solves this problem by augmenting the training set with patterns in which the vehicle appears situated differently relative to the environment.

Another training set restriction results from the limited time available for data collection. Since ALVINN needs to rapidly adapt to new situations, it is impossible to collect data which illustrates all the situations it might eventually encounter. Adding structured noise to the input mitigates this problem by simulating rare situations, such as passing cars, which the network will inevitably encounter when driving autonomously.

A related training set restriction is the incremental nature of the data collection process. In many real world problems, the network cannot expect to have the entire training set available from the start. Instead, training examples become available one at a time. While it might be possible to entirely separate the data collection and the training phases, this would have two drawbacks. First, the time required to create a network capable of performing in a new situation would be increased. Second, it would be difficult to determine when sufficient examples had been collected to train a reliable network, since there is no way to judge the performance of a network before it is trained. Training the network simultaneously with data collection decreases development time. It also allows one to judge the sufficiency of the training set by measuring how closely the network mimics the person's steering response as training progresses. However training "on-the-fly" creates its own difficulties. It introduces the potential for overlearning particular responses in repetitive environments. By intelligently buffering previously encountered examples to ensure variety in the training set, I have shown data collection and training of artificial neural networks can occur simultaneously.

All of these specific extensions to connectionist training procedures are examples of a single important principle. Applying artificial neural networks to difficult real world problems requires exploiting a priori domain knowledge. Previous work in specialized network architectures has demonstrated this principle in the domains of speech [Waibel et al., 1987] and character recognition [LeCun et al., 1989]. For these tasks, it is unrealistic to expect a standard three layer network to perform reliably. This dissertation demonstrates that for certain real world problems, specialized architectures are not necessary. However to achieve reliable performance does require modifying the standard training techniques based on knowledge of the domain.

Another important lesson from this dissertation is the importance of modularity in connectionist architectures. Training a single network to guide a mobile robot in a wide variety of situations would be extremely difficult. But by using the training procedures described above, separate networks can be quickly trained to handle many different circumstances. However with modularity comes the need to integrate multiple networks. I have developed both symbolic and connectionist techniques for arbitrating between multiple networks. The symbolic integration techniques of relevancy and priority arbitration allow high level knowledge and symbolic reasoning to be effectively combined with the perceptual capabilities of artificial neural networks.

Unlike these symbolic techniques, the connectionist multi-network integration methods developed in this dissertation can weight the responses from many networks without relying on detailed symbolic knowledge of the environment. Unlike previous connectionist arbitration methods, Output Appearance Reliability Estimation (OARE) and Input Reconstruction Reliability Estimation (IRRE) can assess the appropriateness of individual networks independently, without requiring a separate gating network. In addition to facilitating modular network construction, these techniques provide a crucial capability, namely the ability to determine when a network is likely to fail. For many real world tasks, including autonomous driving, failures can be catastrophic. OARE and IRRE provide a means for recognizing potentially confusing situations and hence avoiding costly mistakes.

For those more interested in the general applicability of artificial neural networks to machine vision than in the specific connectionist techniques developed in this dissertation, I can not hope to characterize all the circumstances in which connectionist learning methods provide an effective alternative to the traditional hand-coded approach. However there are a number of features of vision-based mobile robot guidance task which make it particularly amenable to connectionist techniques. The most obvious advantage of the mobile robot guidance tasks is the ready availability of a teaching signal. For autonomous driving, the teaching signal came directly from the human driver. In the walking robot example, the teaching signal was also easy to obtain, since the task involved determining other easily measured quantities, namely foot displacement from a target in two dimensions. The same techniques would not be as readily applicable to domains such as controlling individual joints of a robot arm, or controlling the steps in a complex chemical process, since in these tasks the correct response is difficult to determine.

A related advantage for the task of mobile robot guidance, particularly when

individual networks are trained for restricted domain such as highway driving, is that the important attributes for correct performance are the position and orientation of known image features. This contrasts with classification tasks such as speech or character recognition in which the position and orientation of input features must frequently be ignored in favor of higher level characteristics which determine the input's identity. Like other classification tasks, autonomous robot guidance requires combining many local image cues such as the positions of lane markers and road edges. But unlike speech and character recognition where single small features can mean the difference between one class and another, in mobile robot guidance none of the multiple cues is of special importance. Together they usually support a single, consistent interpretation of the input, making the learning task easier. Ambiguity, which is to be avoided at all costs in other classification tasks where there is always a single correct interpretation of the input, can in fact be encouraged in network for autonomous robot guidance. The kind of ambiguity that results when the vehicle encounters an intersection can be combined with symbolic map information to improve the system's performance.

The ability of artificial neural networks to effectively integrate many local input features is the main advantage connectionist techniques have over hand-coded approaches to mobile robot guidance. Previous systems have relied on the judgement and programming of people to determine which input features are important, how to find them, and how to extract from them the correct response. Each of these subtasks is a serious challenge since most of the knowledge people have about driving is not open to introspection. As a result, previous hand-programmed systems have employed little of the information contained in sensor images, and therefore have often suffered from low reliability. Those systems that have been programmed to find enough features in one domain for reliable performance, are usually so situation-specific that they are incapable of handling other situations.

I have demonstrated that connectionist methods can make develop of autonomous navigation systems for specific situations much easier. In addition, this dissertation illustrates that there need not be a direct tradeoff between a vision system's generality and its reliability. Using machine learning techniques, it is possible to build a vision system capable of performing accurately in a wide variety of circumstances.

12.2 Future Work

Notwithstanding its past successes, much work remains to be done on the ALVINN system. To improve its generality, additional networks need to be trained for new circumstances. One near term extension is to train networks to drive using a newly acquired infrared camera. Slight temperature variations between important features in the scene should make it possible to drive using this sensor even in total darkness. Unlike the laser sensor, which also requires no ambient lighting, the infrared camera operates at the same high frame rate as a video camera. This should make it useful for high speed driving.

Another well defined extension is to integrate a satellite positioning system into the symbolic mapping module. While the 10 meter resolution of current satellite positioning technology limits its usefulness for steering the vehicle, it could be employed in conjunction with a map to greatly enhance ALVINN's range. With a satellite positioning system to indicate when the vehicle is approaching an important exit, it should be possible to achieve very long autonomous runs (e.g., Pittsburgh to Chicago).

As the perceptual capabilities of the connectionist components improve, it should be possible to rely less on detailed symbolic knowledge of the environment and more on neural network processing to guide high level decision making. One way to achieve this goal is through more widespread and sophisticated application of the connectionist reliability estimation techniques I have developed. Another is through the development of additional perception modules for tasks like recognizing stop lights or street signs.

Both of these approaches will eventually require a capability not yet present in the ALVINN system: the ability to change the orientation of the sensors to view potentially important aspects of the scene. Currently the system uses fixed sensors pointed directly in front of the vehicle. In order to successfully traverse arbitrary intersections, to say nothing of recognize other environmental cues such as stop lights, will require the ability to pan and tilt the sensors. The Navlab II will soon be equipped with a computer-controlled sensor platform which will make this possible. The question remains how to intelligently control a sensor's field of view, and change a network's focus of attention appropriately.

From a long term applications perspective, it may eventually be possible to employ the techniques developed in this dissertation to implement a "super cruise control" that automates a vehicle's steering as well as speed. In this scenario, a person would drive for a short time to train the network in the current situation.

After training, the network would take over and drive autonomously. More practical opportunities to apply these ideas exist in less demanding domains, such as automated farm equipment, and in situations considered dangerous for people, such as space and undersea exploration. The SM^2 example demonstrates the wide potential for application of these ideas. However much work remains to be done to ensure and demonstrate the reliability of connectionist robot guidance techniques in the harsh conditions characteristic of these alternative domains.

Bibliography

[Amidi & Thorpe, 1990] Amidi, O, and Thorpe, C. (1990) Integrated mobile robot control. *Proceedings SPIE Mobile Robots V, vol. 1388*, pp 504-524.

[Andersen et al., 1985] Andersen, R.A., Essick, G.K. and Siegel, R.M. (1985) Encoding of spatial location by posterior parietal neurons. *Science Vol. 230*, pp 456-458.

[Bachrach, 1991] Bachrach, J.R. (1991) A connectionist learning control architecture for navigation. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 457-463.

[Baldi, 1991] Baldi, P. (1991) Computing with arrays of bell-shaped and sigmoid functions. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 735-742.

[1] [Baldi & Hornik, 1989] Baldi, P. and Hornik, K. (1989) Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks, Vol. 2* pp. 53-58.

[Ballard & Brown, 1982] Ballard, D. and Brown, C. (1982) *Computer Vision*, Prentice Hall Inc.

[Blaauw, 1982] Blaauw, G.J. (1982) Driving experience and task demands in simulator and instrumented car: A validation study, *Human Factors, Vol. 24*, pp. 473-486.

[Breiman et al., 1984] Breiman L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984) *Classification and Regression Trees*, Wadsworth, Belmont.

[Brooks, 1986] Brooks, R.A. (1986) A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 1, pp. 14-23.

[Brown, Friedman & Kanade, 1990] Brown, H.B., Friedman, M.B., and Kanade, T. (1990) A Simple 5-DOF Walking Robot for Space Station Application; *Space Operations, Applications, and Research.*

[Cottrell, 1990] Cottrell, G.W. (1990) Extracting features from faces using compression networks: Face, identity, emotion and gender recognition using holons. *Connectionist Models: Proc. of the 1990 Summer School*, David Touretzky (Ed.), Morgan Kaufmann, San Mateo, CA.

[2] [Cottrell & Munro, 1988] Cottrell, G.W. and Munro, P. (1988) Principal components analysis of images via back-propagation. *Proc. Soc. of Photo-Optical Instr. Eng.*, Cambridge MA.

[Crisman & Thorpe, 1990] Crisman, J.D. and Thorpe, C. (1990) Color Vision for Road Following. *Vision and Navigation: The CMU Navlab* C. Thorpe (Ed.), Kluwer Academic Publishers, Boston.

[Dayan, 1991] Dayan, P. (1991) Navigating through temporal difference. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 464-470.

[Dickmanns & Zapp, 1987] Dickmanns, E.D. and Zapp, A. (1987) Autonomous high speed road vehicle guidance by computer vision. *Proceedings of the 10th World Congress on Automatic Control, Vol. 4*, Munich, West Germany.

[Dickmanns & Zapp, 1986] Dickmanns, E.D., Zapp, A. (1986) A curvature-based scheme for improving road vehicle guidance by computer vision. *"Mobile Robots", SPIE-Proc. Vol. 727*, Cambridge, MA.

[Fahlman & Lebiere, 1990] Fahlman, S.E. and Lebiere, C. (1991) The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (ed.), Morgan Kaufmann, pp. 524-532.

[Fahlman, 1988] Fahlman, S.E. (1988) Faster-learning variations on back-propagation: an empirical study. *Proceedings of the Connectionist Summer School*, Morgan Kaufmann.

[Friedman, Bentley & Finkel, 1977] Friedman, J.H., Bentley, J.L. and Finkel, R.A. (1977) An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software 3:3*, pp. 209-226.

[Fukushima, 1988] Fukushima, K. (1988) A neural network for visual pattern recognition. *IEEE Computer 21:3*, pp. 65-75.

[Goldberg, 1989] Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

[Gorman & Sejnowski, 1988] Gorman, R.P. and Sejnowski, T.J. (1988) Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks Vol. 1*, pp. 75-89.

[Goto et al., 1986] Goto, Y., Matsuzaki, K., Kweon, I., and Obatake, T. (1986) CMU Sidewalk Navigation System: A blackboard-based outdoor navigation system using sensor fusion with colored-range images, *Proceedings of the Joint Computer Conference*.

[Grossberg, 1987] Grossberg, S. (1987) Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11:23-63.

[Gusciora et al., 1990] Gusciora, G.L., Pomerleau, D.A., Touretzky, D.S., Kung, H.T. (1990) Back-propagation on Warp. *Artificial Neural Networks: Applications and Implementations*, Ben Wah, Manoel F. Tenorio, Pankaj Mehra and Jose A.B. Fortes (Eds.) IEEE Computer Society Press.

[Hampshire, 1990] Hampshire, J.B. (1990) A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Transactions on Neural Networks 1:2*, pp 216-228.

[Hampshire & Waibel, 1992] Hampshire J.B. and Waibel, A.H. (1989) The Meta-Pi network: building distributed knowledge representations for robust pattern recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*.

[Hanson, 1990] Hanson, S.J. (1991) Meiosis networks. *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (ed.), Morgan Kaufmann, pp. 533-541.

[Holland, 1975] Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems* University of Michigan Press.

[Hopfield, 1982] Hopfield, J.J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, U.S.A., 79, 2554-2558.

[Hubel & Wiesel, 1979] Hubel, D.H., Wiesel, T.N. (1979) Brain mechanisms of vision, *Scientific American, Vol. 241, No. 3.*

[Jacobs et al., 1990] Jacobs, R.A., Jordan, M.I. and Barto, A.G. (1990) Task decomposition through competition in a modular connectionist architecture: the what and where vision tasks. *Cognitive Science*, in press.

[Jacobs et al., 1991] Jacobs, R.A., Jordan, M.I. Nowlan, S.J. and Hinton, G.E. (1991) Adaptive mixtures of local experts. *Neural Computation, 3:1*, Terrence Sejnowski (ed).

[Jordan, 1988] Jordan, M.I. (1988) Supervised learning and systems with excess degrees of freedom. COINS Tech. Report 88-27, Computer and Information Science, University of Massachusetts, Amherst MA.

[Jordan & Jacobs, 1990] Jordan, M.I. and Jacobs, R.A. (1990) Learning to control an unstable system with forward modeling. *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (ed.), Morgan Kaufmann, pp. 324-331.

[Katayama & Kawato, 1991] Katayama, M., and Kawato M. (1991) Learning trajectory and force control of an artificial muscle arm by parallel-hierarchical neural network model. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 436-442.

[Keesing & Stork, 1991] Keesing, R. and Stork, D.G. (1991) Evolution and learning in neural networks: The number and distribution of learning trials affect the rate of evolution. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 804-810.

[Kenue, 1989] Kenue, S.K. (1989) Lanelok: Detection of lane boundaries and vehicle tracking using image-processing techniques. *SPIE Conference on Aerospace Sensing, Mobile Robots IV*, Nov. 1989.

[Kluge & Thorpe, 1990] Kluge, K. and Thorpe, C. (1990) Explicit models for robot road following. *Vision and Navigation: The CMU Navlab* C. Thorpe (Ed.), Kluwer Academic Publishers, Boston.

[Koch et al., 1990] Koch, C., Bair, W., Harris, J.G., Horiuchi, T., Hsu, A. and Luo, J. (1990) Real-time computer vision and robotics using analog VLSI circuits. *Advances in Neural Information Processing Systems, 2*, D.S. Touretzky (Ed.), Morgan Kaufmann, San Mateo, CA.

[Kohonen, 1990] Kohonen, T. (1990) *Self-Organization and Associative Memory*, 3rd. ed., Springer-Verlag.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L.D. (1989) Back-propagation applied to handwritten zip code recognition. *Neural Computation 1:4*, Terrence Sejnowski (ed.).

[Lehky & Sejnowski, 1988] Lehky, S.R. and Sejnowski, T.J. (1988) Network model of shape-from-shading: neural function arises from both receptive and projective fields. *Nature, Vol. 333 No. 2*, pp. 452-454.

[Linden & Kindermann, 1989] Linden, A., and Kindermann, J. (1989) Inversion of multilayer nets. *Proceedings of IEEE International Joint Conference on Neural Networks, Vol. II*, pp. 425-430.

[3] [Linsker, 1988] Linsker, R. (1988) Self-organization in a perceptual network", *Computer, Vol. 21* pp. 105-117.

[Linsker, 1989] Linsker, R. (1989) Designing a sensory processing system: What can be learned from principal component analysis? IBM Technical Report RC14983 (#66896).

[MacQueen, 1967] MacQueen, J. (1967) Some methods of classification and analysis of multivariate observations. In L.M. LeCam and J. Neyman, editors, *Proc. 5th Berkeley Symposium on Math., Stat., and Prob.*, U. California Press.

[Marra, Dunlay & Mathis, 1988] Marra, M., Dunlay, T.R., Mathis, D. (1988) Terrain classification using texture for the ALV. Martin Marietta Information and Communications Systems technical report 1007-10.

[Moody & Darken, 1989] Moody, J. and Darken, C.J. (1989) Fast learning in networks of locally-tuned processing units. *Neural Computation* 1:281-294.

[Oja, 1989] Oja, E. (1989) Neural networks, principal components and subspaces. *International Journal of Neural Systems*, 1(1):61-68.

[O'Shaughnessy, 1987] O'Shaughnessy, D. (1987) *Speech Communication: Human and Machine*, Addison-Wesley.

[Pearlmutter, 1988] Pearlmutter, B. (1988) Learning state space trajectories in recurrent neural networks. *Proceedings of the Connectionist Summer School*, Morgan Kaufmann.

[Pineda, 1987] Pineda, F. (1987) Generalization of back-propagation to recurrent neural networks. *Physical Review Letters, Vol. 19, No. 59*, pp. 2229-2232.

[Poggio & Girosi, 1990] Poggio, T., and Girosi, F. (1990) Regularization algorithms for learning that are equivalent to multilayer networks. *Science, Vol. 247* pp. 987-982.

[Pomerleau, in press] Pomerleau, D.A. (in press) Neural Network Based Vision for Precise Control of a Walking Robot. To appear in a special issue of *Machine Learning*. Alex Waibel (Ed.).

[Pomerleau, 1991a] Pomerleau, D.A. (1991) Efficient Training of Artificial Neural Networks for Autonomous Navigation. *Neural Computation 3:1*, Terrence Sejnowski (ed).

[Pomerleau, 1991b] Pomerleau, D.A. (1991) Neural network-based vision processing for autonomous robot guidance. *Proceedings of SPIE Conference on Aerospace Sensing*, Orlando, Fl.

[Pomerleau, 1991c] Pomerleau, D.A. (1991) Rapidly Adapting Artificial Neural Networks for Autonomous Navigation. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 429-435.

[Pomerleau et al., 1991d] Pomerleau, D.A., Gowdy, J., Thorpe, C.E. (1991) Combining artificial neural networks and symbolic processing for autonomous robot guidance. *Engineering Applications of Artificial Intelligence, 4:4* pp. 279-285.

[Pomerleau, 1990] Pomerleau, D.A. (1990) Neural network based autonomous navigation. *Vision and Navigation: The CMU Navlab* C Thorpe (Ed.), Kluwer Academic Publishers.

[Pomerleau, 1989] Pomerleau, D.A. (1989) ALVINN: An Autonomous Land Vehicle In a Neural Network. *Advances in Neural Information Processing Systems 1*, D.S. Touretzky (ed.), Morgan Kaufmann.

[Pomerleau et al., 1988] Pomerleau, D.A., Gusciora, G.L., Touretzky, D.S., and Kung, H.T. (1988) Neural network simulation at Warp speed: How we got 17 million connections per second. *Proceedings of IEEE International Joint Conference on Neural Networks*, San Diego, CA.

[Quinlan, 1986] Quinlan, J.R. (1986) Induction of decision trees. *Machine Learning Vol. 1*, pp. 81-106.

[Rohwer, 1991] Rohwer, R. (1991) Time trials on second-order and variable-learning-rate algorithms. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 977-983.

[Reid, Solowka & Billing, 1981] Reid, L.D., Solowka, E.N., and Billing, A.M. (1981) A systematic study of driver steering behaviour. *Ergonomics Vol. 24*, pp. 447-462.

[Rosenblatt & Payton, 1989] Rosenblatt, J.K., Payton, D.W. (1989) A fine-grained alternative to the subsumption architecture for mobile robots. *Proceedings of IEEE International Joint Conference on Neural Networks*, Washington, DC.

[Rumelhart, Hinton & Williams, 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations* pp. 318-362, Bradford Books/MIT Press, Cambridge, MA.

[Rumelhart & Zipser, 1986] Rumelhart, D.E. and Zipser, D. (1986) Feature discovery by competitive learning. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations* pp. 318-362, Bradford Books/MIT Press, Cambridge, MA.

[Seitzma & Dow, 1991] Seitzma, J., and Dow, R. (1991) Creating Artificial Neural Networks that Generalize. *Neural Networks, Vol. 4* pp. 67-79, Pergamon Press.

[Servan-Schreiber et al., 1989] Servan-Schreiber D., Cleeremans, A., and McClelland J.L. (1989) Learning sequential structure in simple recurrent networks, *Advances in Neural Information Processing Systems 1*, D.S. Touretzky (ed.), Morgan Kaufmann.

[Shepanski & Macy, 1987] Shepanski, J.F., Macy, S.A. (1987) Manual training techniques of autonomous systems based on artificial neural networks. *Proceedings of IEEE International Joint Conference on Neural Networks*, San Diego, CA.

[Simon, 1990] Simon, D.A. (1990) Personal communications.

[Sobel, 1970] Sobel, I. (1970) Camera models and machine perception. Stanford AI Lab Technical Report AIM-21.

[Sutton, 1990] Sutton R.S. (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proc. of the Seventh Int. Conf. on Machine Learning*, Morgan Kaufmann.

[Tarassenko et al., 1991] Tarassenko, L., Brownlow, M., Gillian M., Tombs, J. (1991) Real-time autonomous robot navigation using VLSI neural networks. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 422-428.

[Thorpe, 1989] Thorpe, C. (1989) Outdoor visual navigation for autonomous robots. *Proceedings of Intelligent Autonomous Systems 2*. Amsterdam.

[Thorpe et al., 1991] Thorpe C., Amidi O., Gowdy J., Hebert M., Pomerleau D. (1991) Integrating position measurement and image understanding for

autonomous vehicle navigation. To appear in *Proceedings of the Second International Workshop on High Precision Navigation*, Stuttgart, Germany.

[Thorpe & Gowdy, 1990] Thorpe, C. and Gowdy, J. (1990) Annotated Maps for Autonomous Land Vehicles. *Proceedings of the DARPA Image Understanding Workshop*.

[Thrun, 1991] Thrun, S., Moller, K., Linden, A. (1991) Planning with an adaptive world model. *Advances in Neural Information Processing Systems 3*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky (ed.), Morgan Kaufmann, pp. 450-456.

[Touretzky & Pomerleau, 1989] Touretzky, D.S., Pomerleau, D.A. (1989) What's hidden in the hidden units? *BYTE 14:8*, August, 1989.

[Turk et al., 1988] Turk, M. A., Morgenthaler, D. G., Gremban, K. D., Marra, M. (1988) VITS – A vision system for autonomous land vehicle navigation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10.*

[Ueno, Xu & Brown, 1990] Ueno, H., Xu, Y. and Brown, H.B. (1990) On Control and Planning of a Space Station Robot Walker. *Proc. of 1990 IEEE International Conference on Systems Engineering*, Pittsburgh, PA.

[Waibel et al., 1987] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang K. (1987) Phoneme recognition using time-delay neural networks. ATR Technical Report TR-I-0006.

[Waibel, 1989] Waibel, A. (1989) Modular Construction of Time Delay Neural Networks for Speech Recognition. *Neural Computation 1:1*, Terrence Sejnowski (ed.)

[Wallace et al., 1985a] Wallace, R., Matsuzaki, K., Goto, Y., Crisman, J., Webb, J., and Kanade, T. (1985) Progress in robot road-following. Autonomous Mobile Robots Annual Report 1985, Carnegie Mellon University Robotics Institute Technical Report CMU-RI-TR-86-4.

[Wallace et al., 1985b] Wallace, R., Stentz, A., Thorpe, C., Moravec, H., Whittaker, W., Kanade, T. (1985) First results in robot road-following. *Proceedings of Int. Joint Conf. on Artificial Intelligence.*

[Wright, 1989] Wright, W.A. (1989) Contextual road finding with a neural network. British Aerospace Advanced Information Processing Department technical report.

[Zipser & Anderson, 1988] Zipser D. and Andersen R.A. (1988) A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature, Vol. 333, No. 2*, pp. 679-683.